NEW

Coding for Beginners

Everything you need to start learning how to code

60 FREE Programs inside!

100% INDEPENDENT

Featuring
C++, Python, Linux Scripting and more

Packed
with top coding tips and tutorials

Discover
the essential tools you need to code

VISIT…

LANZAROTE
Caliente.COM

Black Dog Media

Master Your Tech

From Beginner to Expert

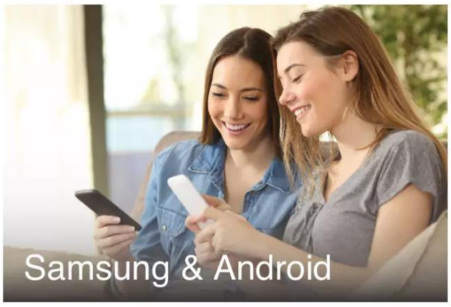
To continue learning more about your tech visit us at:

www.bdmpublications.com

FREE Tech Guides

Apple iPhone, iPad, Mac, MacBook & Watch

PC & Windows 10


Samsung & Android

Photography, Photoshop & Lightroom

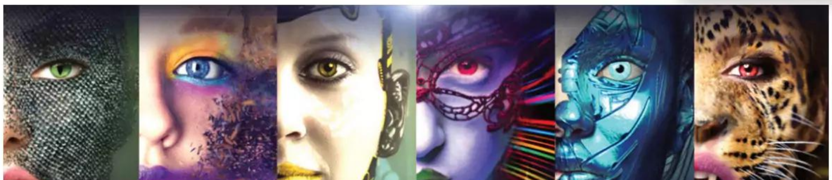
Coding Python, Raspberry Pi & Linux

EXCLUSIVE Offers on our Tech Guidebooks

- Print & digital editions
- Featuring the very latest updates
- Step-by-step tutorials and guides
- Created by BDM experts

Check out our latest titles today!

The iOS 14 iPhone GuideBook
Raspberry Pi Tips, Tricks & Fixes
Coding for Beginners
The python Coding Manual
The iPad GuideBook 14
Windows 10 The Complete Guide
Photography The Beginners' Guide
The Adobe Photoshop Manual


BDM's *Ultimate* Photoshop

bdmpublications.com/ultimate-photoshop

Buy our Photoshop guides and download tutorial images for free! *Simply sign up and get creative.*

PLUS

SPECIAL DEALS and Bonus Content

Sign up to our monthly newsletter and get the latest updates, offers and news from BDM. We are here to help you Master Your Tech!

Coding
for Beginners

Discover how to turn your ideas into code!

Starting to learn to code isn't easy but we're here to help you get going. In the pages of this book you can learn how to start coding using Python, C++, Linux scripting and Windows batch files. We also cover the common pitfalls and mistakes every coder falls into and ways to avoid them in the future; also where you can find help and how to experiment with your code.

We've put together a great collection of tutorials and step-by-step guides to help you understand how to start coding and what you need to turn your ideas into ones and zeros that will help you become a better coder. Learning how to code is an on-going experience, where you learn something new every day and every time you run your code.

Come with us and let's start coding!

Contents

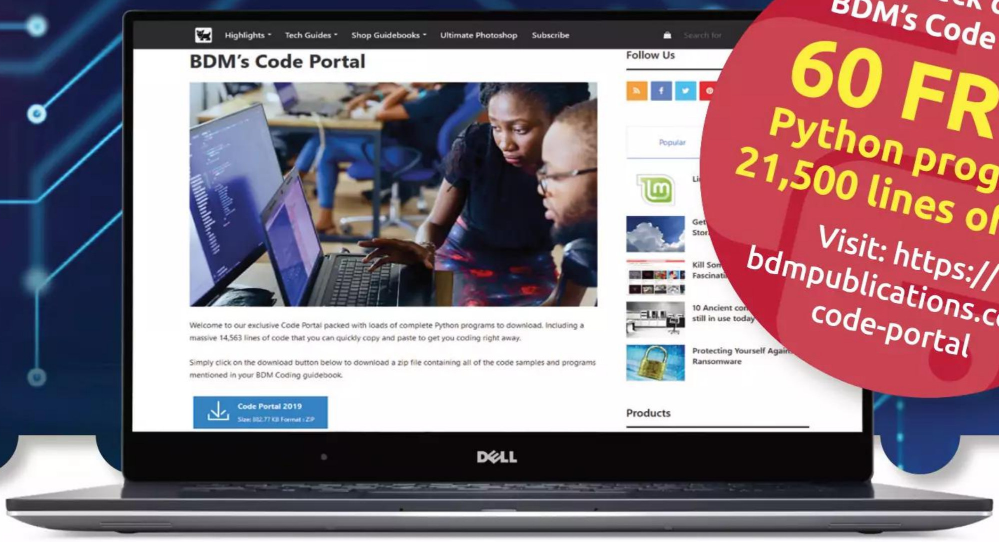
6 Coding Foundations
8 Being a Programmer
10 A Brief History of Coding
12 Choosing a Programming Language
14 Creating a Coding Platform

16 Introducing C++
18 Why C++?
20 C++ Facts & Figures
22 Equipment You Will Need
24 Structure of C++ Code
26 How to Set Up C++ in Windows
28 How to Set Up C++ on a Mac
30 How to Set Up C++ in Linux
32 Other C++ IDEs to Install

34 Coding with C++
36 Your First C++ Program
38 Compile and Execute
40 Using Comments
42 Variables
44 Data Types
46 Strings
48 C++ Maths
50 User Interaction

52 Introducing Python
54 Why Python?
56 What Can You Do with Python?
58 Python in Numbers
60 Equipment You Will Need
62 Getting to Know Python
64 How to Set Up Python in Windows
66 How to Set Up Python in Linux
68 Python on the Pi

70 Coding with Python
72 Starting Python for the First Time
74 Your First Code
76 Saving and Executing Your Code
78 Executing Code from the Command Line
80 Numbers and Expressions
82 Using Comments



BDM's Code Portal

Welcome to our exclusive Code Portal packed with loads of complete Python programs to download. Including a massive 14,563 lines of code that you can quickly copy and paste to get you coding right away.

Simply click on the download button below to download a zip file containing all of the code samples and programs mentioned in your BDM Coding guidebook.

Code Portal 2019
Size: 882.77 KB Format : ZIP

Check out
BDM's Code Portal
60 FREE
Python programs
21,500 lines of code
Visit: https://
bdmpublications.com/
code-portal

Master Python with the help of our fantastic Code Portal, featuring code for games, tools and more.

84 Working with Variables
86 User Input
88 Creating Functions
90 Conditions and Loops
92 Python Modules

94 Coding with Windows 10 Batch Files

96 What is a Batch File?
98 Getting Started with Batch Files
100 Getting an Output
102 Playing with Variables
104 Batch File Programming
106 Loops and Repetition
108 Creating a Batch File Game

110 Coding on Linux

112 Why Linux?
114 The Best Linux Distributions
116 Equipment You Will Need
118 Creating a Linux Installer on Windows
120 Installing Linux on a PC
122 Installing a Virtual Environment
124 Installing Linux in a Virtual Environment
126 Getting Ready to Code in Linux
128 Creating Bash Scripts – Part 1
130 Creating Bash Scripts – Part 2
132 Creating Bash Scripts – Part 3
134 Creating Bash Scripts – Part 4
136 Creating Bash Scripts – Part 5
138 Command Line Quick Reference

140 Code Projects and Ideas

142 Planning your Code
144 Finding Resources
146 Creating a Loading Screen
148 Planning a Text Adventure
150 Text Adventure Script
152 Retro Coding
156 Common Coding Mistakes
158 Python Beginner's Mistakes
160 C++ Beginner's Mistakes

Everything you do online, every time you power on your smart TV, whenever you use your in-car GPS, use your phone, play a game on a tablet, console or PC, it's all been coded by a group of people. All these ones and zeros are developed by those who have learned how to code, and with this book you too can learn how to get started on the road to becoming a programmer.

We cover C++, Python, and Linux Scripting within these pages. There's a huge section of project ideas, type-in listings, and in-depth looks at how code works. But first, let's begin with a good foundation.

8 Being a Programmer
10 A Brief History of Coding
12 Choosing a Programming Language
14 Creating a Coding Platform

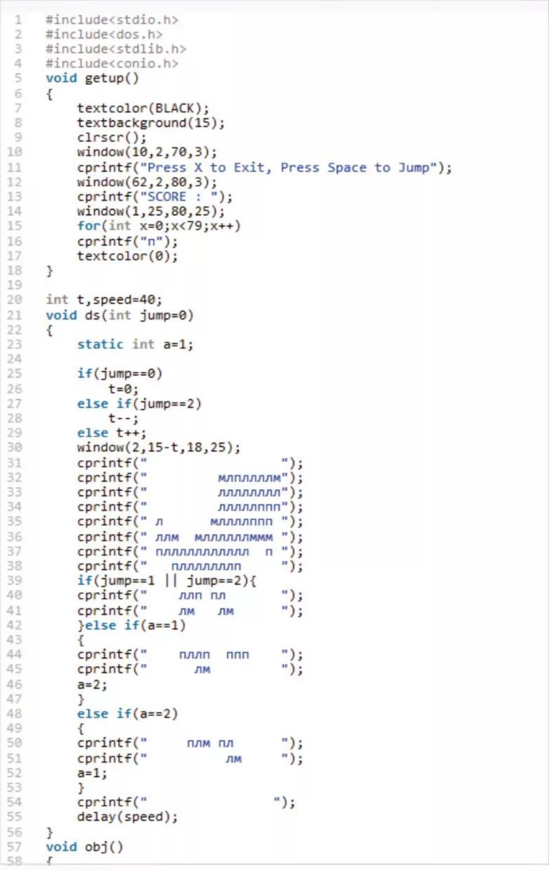
Welcome, Future Coder

Being a Programmer

Programmer, developer, coder, they're all titles for the same occupation, someone who creates code. What they're creating the code for can be anything from a video game to a critical element on board the International Space Station. How do you become a programmer though?

Times have changed since programming in the '80s, but the core values still remain.

> "It's up to you how far to take your coding adventure!"



```
#include<stdio.h>
#include<dos.h>
#include<stdlib.h>
#include<conio.h>
void getup()
{
    textcolor(BLACK);
    textbackground(15);
    clrscr();
    window(10,2,70,3);
    cprintf("Press X to Exit, Press Space to Jump");
    window(62,2,80,3);
    cprintf("SCORE : ");
    window(1,25,80,25);
    for(int x=0;x<79;x++)
    cprintf("n");
    textcolor(0);
}

int t,speed=40;
void ds(int jump=0)
{
    static int a=1;

    if(jump==0)
        t=0;
    else if(jump==2)
        t--;
    else t++;
    window(2,15-t,18,25);
    cprintf("                ");
    cprintf("        млпллллм");
    cprintf("        лллллллл");
    cprintf("        лллллппп");
    cprintf(" л     мллллппп ");
    cprintf(" ллм  мллллллммм ");
    cprintf(" пплллллллллл  п ");
    cprintf("   пплллллллп     ");
    if(jump==1 || jump==2){
    cprintf("    ллп пл      ");
    cprintf("    лм   лм      ");
    }else if(a==1)
    {
    cprintf("    пллп  ппп    ");
    cprintf("      лм        ");
    a=2;
    }
    else if(a==2)
    {
    cprintf("     плм пл      ");
    cprintf("          лм     ");
    a=1;
    }
    cprintf("               ");
    delay(speed);
}
void obj()
{
```

Being able to follow a logical pattern and see an end result is one of the most valued skills of a programmer.

MORE THAN CODE

For those of you old enough to remember the '80s, the golden era of home computing, the world of computing was a very different scene to how it is today. 8-bit computers that you could purchase as a whole, as opposed to being in kit form and you having to solder the parts together, were the stuff of dreams; and getting your hands on one was sheer bliss contained within a large plastic box. However, it wasn't so much the new technology that computers then offered, moreover it was the fact that for the first time ever, you could control what was being viewed on the 'television'.

Instead of simply playing one of the thousands of games available at the time, many users decided they wanted to create their own content, their own games; or simply something that could help them with their homework or home finances. The simplicity of the 8-bit home computer meant that creating something from a few lines of BASIC code was achievable and so the first generation of home-bred programmer was born.

From that point on, programming expanded exponentially. It wasn't long before the bedroom coder was a thing of the past and huge teams of designers, coders, artists and musicians were involved in making a single game. This of course led to the programmer becoming more than simply someone who could fashion a sprite on the screen and make it move at the press of a key.

Naturally, time has moved on and with it the technology that we use. However, the fundamentals of programming remain the same; but what exactly does it take to be a programmer?

The single most common trait of any programmer, regardless of what they're doing, is the ability to see a logical pattern. By this we mean someone who can logically follow something from start to finish and envisage the intended outcome. While you may not feel you're such a person, it is possible to train your brain into this way of thinking. Yes, it takes time but once you start to think in this particular way you will be able to construct and follow code.

Second to logic is an understanding of mathematics. You don't have to be at a genius level but you do need to understand the rudiments of maths. Maths is all about being able to solve a problem and code mostly falls under the umbrella of mathematics.

Being able to see the big picture is certainly beneficial for the modern programmer. Undoubtedly, as a programmer, you will be part of a team of other programmers, and more than likely part of an even bigger team of designers, all of whom are creating a final product. While you may only be expected to create a small element of that final product, being able to understand what everyone else is doing will help you create something that's ultimately better than simply being locked in your own coding cubicle.

Finally, there's also a level of creativity needed to be a good programmer. Again though, you don't need to be a creative genius, just have the imagination to be able to see the end product and how the user will interact with it.

There is of course a lot more involved in being a programmer, including learning the actual code itself. However, with time, patience and the determination to learn, anyone can become a programmer. Whether you want to be part of a triple-A video game team or simply create an automated routine to make your computing life easier, it's up to you how far to take your coding adventure!

A Brief History of Coding

It's easy to think that programming a machine to automate a process or calculate a value is a modern concept that's only really happened in the last fifty years or so. However, that assumption is quite wrong, coding has actually been around for quite some time.

01000011 01101111 01100100 01100101

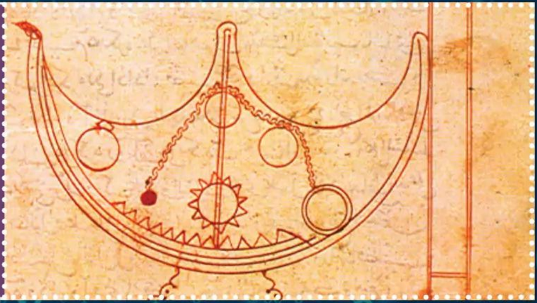
Essentially all forms of coding are made up of ones and zeros, on or off states. This works for a modern computer and even the oldest known computational device.

~87 BC

It's difficult to pinpoint an exact start of when humans began to 'program' a device. However, it's widely accepted that the Antikythera Mechanism is possibly the first 'coded' artefact. It's dated to about 87 BC and is an ancient Greek analogue computer and orrery used to predict astronomical positions.

~850 AD

The Banū Mūsā brothers, three Persian scholars who worked in the House of Wisdom in Baghdad, published the Book of Ingenious Devices in around 850 AD. Among the inventions listed was a mechanical musical instrument, a hydro-powered organ that played interchangeable cylinders automatically.



1800

Joseph Marie Jacquard invents a programmable loom, which used cards with punched holes to create the textile design. However, it is thought that he based his design on a previous automated weaving process from 1725, by Basile Bouchon.

1842-1843

Ada Lovelace translated the memoirs of the Italian mathematician, Francis Maneclang, regarding Charles Babbage's Analytical Engine. She made copious notes within her writing, detailing a method of calculating Bernoulli Numbers using the engine. This is recognised as the first computer program. Not bad, considering there were no computers available at the time.

1930-1950

During the Second World War, there significant advances were made in programmable machines. Most notably, the cryptographic machines used to decipher military codes used by the Nazis. The Enigma was invented by the German engineer Arthur Scherbius but was made famous by Alan Turing at Bletchley Park's codebreaking centre.

From the 1970s, the development of the likes of C, SQL, C with Classes (C++), MATLAB, Common Lisp and more came to the fore. The '80s was undoubtedly the golden age of the home computer, a time when silicon processors were cheap enough for ordinary folk to buy. This led to a boom in home/bedroom coders with the rise of 8-bit machines.

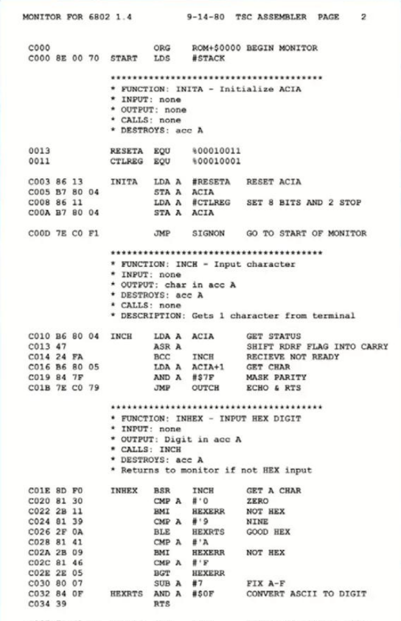
1951-1958

1959

1960-1970

1970-1985

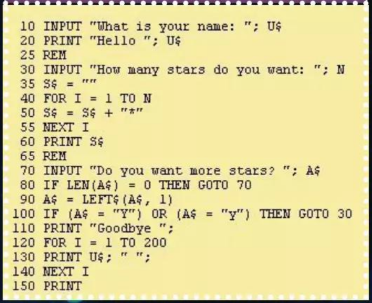
1990s-Present Day



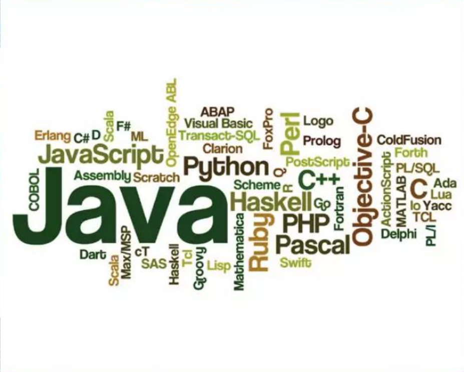
```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

 C000                    ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START    LDS    #STACK

                ****************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA   EQU    %00010011
 0011           CTLREG   EQU    %00010001

 C003 86 13     INITA    LDA A  #RESETA   RESET ACIA
 C005 B7 80 04           STA A  ACIA
 C008 86 11              LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04           STA A  ACIA

 C00D 7E C0 F1           JMP    SIGNON    GO TO START OF MONITOR

                ****************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH     LDA A  ACIA      GET STATUS
 C013 47                 ASR A            SHIFT RDRF FLAG INTO CARRY
 C014 24 FA              BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05           LDA A  ACIA+1    GET CHAR
 C019 84 7F              AND A  #$7F      MASK PARITY
 C01B 7E C0 79           JMP    OUTCH     ECHO & RTS

                ****************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX    BSR    INCH      GET A CHAR
 C020 81 30              CMP A  #'0       ZERO
 C022 2B 11              BMI    HEXERR    NOT HEX
 C024 81 39              CMP A  #'9       NINE
 C026 2F 0A              BLE    HEXRTS    GOOD HEX
 C028 81 41              CMP A  #'A
 C02A 2B 09              BMI    HEXERR    NOT HEX
 C02C 81 46              CMP A  #'F
 C02E 2E 05              BGT    HEXERR
 C030 80 07              SUB A  #7        FIX A-F
 C032 84 0F     HEXRTS   AND A  #$0F      CONVERT ASCII TO DIGIT
 C034 39                 RTS

 C035 7E C0 AF  HEXERR   JMP    CTRL      RETURN TO CONTROL LOOP
```

The first true computer code was Assembly Language (ASM) or Regional Assembly Language. ASM was specific to the architecture of the machine it was being used on. In 1951 programming languages fell under the generic term Autocode. Soon languages such as IPL, FORTRAN and ALGOL 58 were developed.

Computer programming was mainly utilised by universities, the military and big corporations during the '60s and the '70s. A notable step toward a more user-friendly, or home user language, was the development of BASIC (Beginners All-purpose Symbolic Instruction Code) in the mid-sixties.



```
 10 INPUT "What is your name: "; U$
 20 PRINT "Hello "; U$
 25 REM
 30 INPUT "How many stars do you want: "; N
 35 S$ = ""
 40 FOR I = 1 TO N
 50 S$ = S$ + "*"
 55 NEXT I
 60 PRINT S$
 65 REM
 70 INPUT "Do you want more stars? "; A$
 80 IF LEN(A$) = 0 THEN GOTO 70
 90 A$ = LEFT$(A$, 1)
100 IF (A$ = "Y") OR (A$ = "y") THEN GOTO 30
110 PRINT "Goodbye ";
120 FOR I = 1 TO 200
130 PRINT U$; " ";
140 NEXT I
150 PRINT
```



The Internet age brought a wealth of new programming languages and allowed people access to the tools and knowledge needed to learn coding in a better way. Not only could a user learn how to code but they could freely share their code and source other code to improve their own.

Admiral Grace Hopper was part of the team that developed the UNIVAC I computer and she eventually developed a compiler for it. In time, the compiler she developed became COBOL (Common Business-oriented Language), a computer language that's still in use today.

Choosing a Programming Language

It would be impossible to properly explain every programming language in a single book of this size. New languages and ways in which to 'talk' to a computer or device and set it instructions are being invented almost daily; and with the onset of quantum computing, even more complex methods are being born. Here is a list of the more common languages along with their key features.

java
html
sql
ph
delphi
SQL
JS
C#
C++
Java
Perl Programming
python

SQL SQL stands for Structured Query Language. SQL is a standard language for accessing and manipulating databases. Although SQL is an ANSI (American National Standards Institute) standard, there are different versions of the SQL language. However, to be compliant, they all support at least the major commands such as Select, Update and Delete in a similar manner.

JAVASCRIPT JavaScript (often shortened to JS) is a lightweight, interpreted, object-oriented language with first class functions. JavaScript runs on the client side of the web, that can be used to design or program how the web pages behave on the occurrence of an event. JavaScript is an easy to learn and also powerful scripting language, widely used for controlling web page behaviour.

JAVA Java is the foundation for virtually every type of networked application and is the global standard for developing enterprise software, web-based content, games and mobile apps. The two main components of the Java platform are the Java Application Programming Interface (API) and the Java Virtual Machine (JVM) that translates Java code into machine language.

C# C# is an elegant object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. You can use C# to create Windows client applications, XML Web services, client server applications, database applications and much more. The curly-brace syntax of C# will be instantly recognisable to anyone familiar with C, C++ or Java.

PYTHON Python is a widely used high level programming language used for general purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy that emphasises code readability and a syntax that allows programmers to express concepts in fewer lines of code. This can make it easier for new programmers to learn.

C++ C++ (pronounced cee plus plus) is a general purpose programming language. It has imperative, object-oriented and generic programming features. It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights.

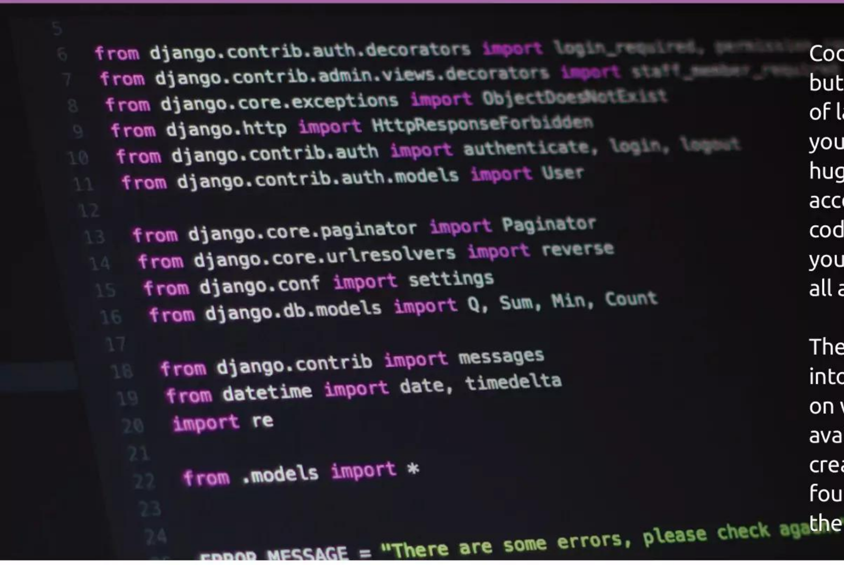
RUBY Ruby is a language of careful balance. Its creator, Yukihiro “Matz” Matsumoto, blended parts of his favourite languages (Perl, Smalltalk, Eiffel, Ada and Lisp) to form a new language. From its release in 1995, Ruby has drawn devoted coders worldwide. Ruby is seen as a flexible language; essential parts of Ruby can be removed or redefined, at will. Existing parts can be added to.

PERL Perl is a general purpose programming language, used for a wide range of tasks including system administration, web development, network programming, GUI development and more. Its major features are that it’s easy to use, supports both procedural and object-oriented (OO) programming, has powerful built-in support for text processing and has one of the most impressive collections of third-party modules.

SWIFT Swift is a powerful and intuitive programming language for macOS, iOS, watchOS and tvOS. Writing Swift code is interactive and fun; the syntax is concise yet expressive and Swift includes modern features that developers love. Swift code is safe by design, yet also produces software that runs lightning fast. A coding tutorial app, Swift Playgrounds, is available for the iPad.

Creating a Coding Platform

The term 'Coding Platform' can signify: a type of hardware on which you can code, a particular operating system, or even a custom environment that's pre-built and designed for easy game creation. In truth it's quite a loose term, as a Coding Platform can be a mixture of all of these ingredients, it's simply down to which programming language you intend to code in and what your end goals are.



```
from django.contrib.auth.decorators import login_required,
 from django.contrib.admin.views.decorators import staff_member_required
  from django.core.exceptions import ObjectDoesNotExist
   from django.http import HttpResponseForbidden
    from django.contrib.auth import authenticate, login, logout
     from django.contrib.auth.models import User

      from django.core.paginator import Paginator
       from django.core.urlresolvers import reverse
        from django.conf import settings
         from django.db.models import Q, Sum, Min, Count

          from django.contrib import messages
           from datetime import date, timedelta
            import re

             from .models import *


               ERROR_MESSAGE = "There are some errors, please check aga
```

Coding can be one of those experiences that sounds fantastic, but is often confusing to tackle. After all, there's a plethora of languages to choose from, countless apps that will enable you to code in a specific, or range, of languages and an equally huge amount of third-party software to consider. In addition, by accessing the Internet, you will discover that there are countless coding tutorials available for the language in which you've decided you want to program, alongside even more examples of code. It's all a little too much at first.

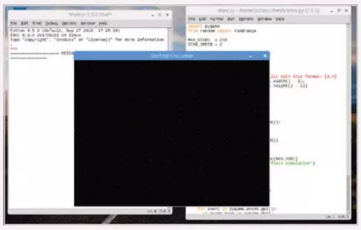
The trick is to slow down and, to begin with, not look too deeply into coding. Like all good projects, you need a solid foundation on which to build your skillset and to have all the necessary tools available to enable you to complete the basic steps. This is where creating a coding platform comes in, as it will be your learning foundation while you begin to take your first tentative steps into the wider world of coding.

HARDWARE

Thankfully, coding at the foundation level doesn't require specialist equipment, or a top of the range, liquid hydrogen-cooled PC. If you own a computer, no matter how basic, you can begin to learn how to code. Naturally, if the computer in question is a Commodore 64 then you may have some difficulty following a modern language tutorial, but some of the best programmers around today started on an 8-bit machine, so there's hope yet.

You will need access to the Internet to download, install and update the coding development environment, alongside a computer with Windows 10, macOS, or Linux, installed. You can use other operating systems, but these are the 'big three' and you will find that most code resources are written with one, or all, of these in mind.

SOFTWARE



In terms of software, most of the development environments have the tools that allow you to code, compile the code and execute it, freely available to download and install. There are some specialist tools available that will cost, but at this level they're not necessary, so don't be fooled into thinking you need to purchase any extra software in order to start learning how to code.

Over time, you may find yourself progressing from the mainstream development environment and using a collection of your own, discovered, tools to write your code. It's all personal preference in the end and as you become more experienced, you will start to use different tools to get the job done.

OPERATING SYSTEMS

Windows 10 is the most widely used operating system in the world, so it's natural that the vast majority of coding tools are written for Microsoft's leading operating system. However, don't discount macOS and especially Linux.

macOS users enjoy an equal number of coding tools to their Windows counterparts. In fact, you will probably find that a lot of professional coders use a Mac over a PC, simply because of the fact that the Mac operating system is built on top of Unix (the command-line OS that powers much of the world's filesystems and servers). This Unix layer lets you test programs in almost any language without using a specialised IDE.

However, Linux is, by far and away, one of the most popular and important coding operating systems available. Not only does it have a Unix-like backbone, it's also free to download, install and use and comes with most of the tools necessary to start learning how to code. Linux powers most of the servers that make up the Internet. It's used on nearly all of the top supercomputers, as well as specifically in organisations such as NASA, CERN and the military, it also forms the base of Android-powered devices, smart TVs and in-car systems. Linux, as a coding platform, is an excellent idea and it can be installed inside a virtual machine without ever affecting the installation of Windows or macOS.

THE RASPBERRY PI

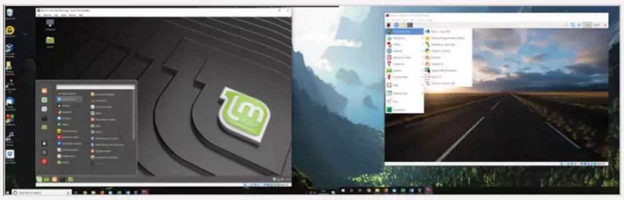
If you haven't already heard of the Raspberry Pi, then we suggest you head over to **www.raspberrypi.org** and check it out. In short, the Raspberry Pi is a small, fully functional computer. It comes with its own customised, Linux-based operating system that's pre-installed with everything you need to start learning how to code in Python, C++, Scratch and more.

Costing around £35, it's incredibly cheap and allows you to utilise different hardware, in the form of robotics and electronics projects, as well as offering a complete desktop experience. Although not the most powerful computing device in the world, the Raspberry Pi has a lot going for it, especially in terms of being one of the best coding platforms available.

VIRTUAL MACHINES

A virtual machine is a piece of software that allows you to install a fully working operating system within the confines of the software itself. The installed OS will allocate user-defined resources from the host computer, providing memory, hard drive space etc., as well as sharing the host computer's Internet connection.

The advantage of a virtual machine is that you can work with Linux, for example, without it affecting your currently installed host OS. This means that you can have Windows 10 running and launch your virtual machine client, boot into Linux and use all the functionality of Linux, while still being able to use Windows.

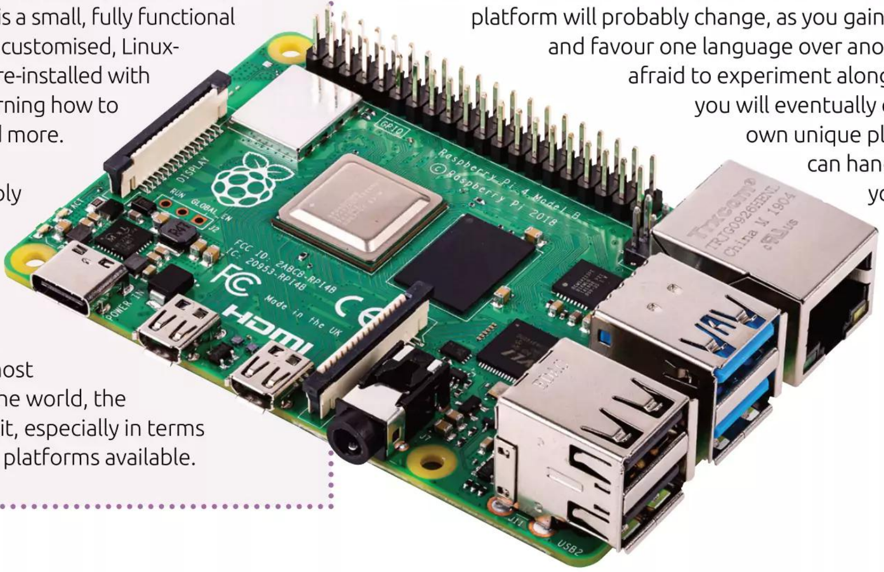


This, of course, makes it a fantastic coding platform, as you can have different installations of operating systems running from the host computer while using different coding languages. You can test your code without fear of breaking your host OS and it's easy to return to a previous configuration without the need to reinstall everything again.

Virtualisation is the key to most big companies now. You will probably find, rather than having a single server with an installation of Windows Server, for example, the IT team have instead opted for a virtualised environment whereby each Windows Server instance is a virtual machine running from several powerful machines. This cuts down on the number of physical machines, allows the team to better manage resources and enables them to deploy an entire server dedicated to a particular task in a fraction of the time.

YOUR OWN CODING PLATFORM

Whichever method you choose, remember that your coding platform will probably change, as you gain experience and favour one language over another. Don't be afraid to experiment along the way, as you will eventually create your own unique platform that can handle all the code you enter into it.

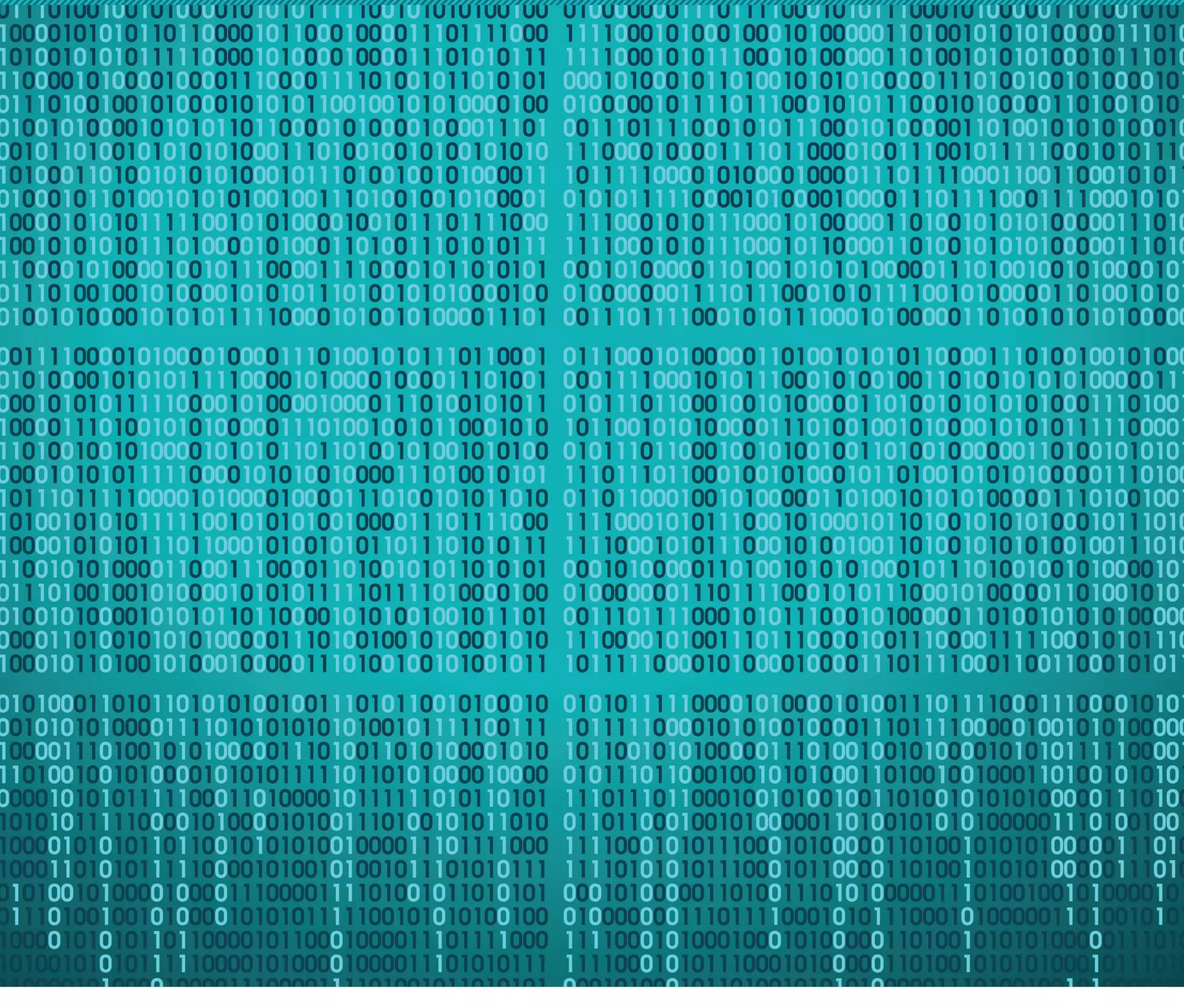


C++ is an amazing programming language. Most of what you see in front of you when you power up your computer, regardless of whether you're using Windows, macOS or Linux, is created using C++. Being able to code in C++ will open a whole new world for you, in terms of desirable professional skills and the ability to code amazing apps and games.

C++ is an efficient and powerful language that's used to develop operating systems, applications, games and much more. It's used in science, engineering, banking, education, the space industry, and much more.

18 Why C++?
20 C++ Facts & Figures
22 Equipment You Will Need
24 Structure of C++ Code
26 How to Set Up C++ in Windows
28 How to Set Up C++ on a Mac
30 How to Set Up C++ in Linux
32 Other C++ IDEs to Install

Introducing C++



Why C++?

C++ is one of the most popular programming languages available today. Originally called C with Classes, the language was renamed to C++ in 1983. It's an extension of the original C language, and is a general purpose object-oriented (OOP) environment.

C EVERYTHING

Due to both the complexity of the language and its power and performance, C++ is often used to develop games, programs, device drivers, and even entire operating systems.

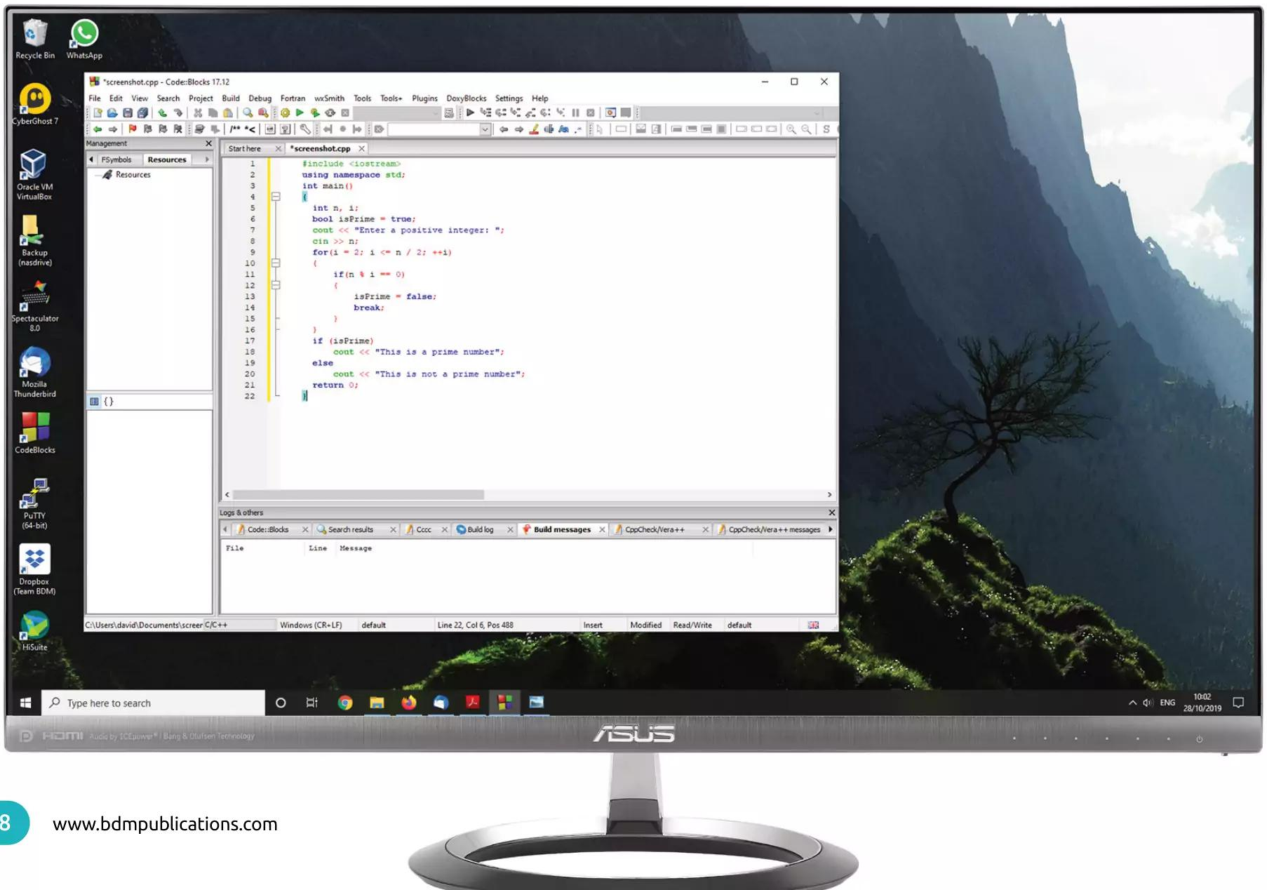
Dating back to 1979, the start of the golden era of home computing, C++, or rather C with Classes, was the brainchild of Danish computer scientist Bjarne Stroustrup, while working on his Ph.D. thesis. Stroustrup's plan was to further the original C language, which had been widely used since the early seventies.

C++ proved to be popular among the developers of the 80s, since it was a much easier environment with which to get to grips, and, more importantly, it was 99% compatible with the original C language. This meant that, beyond the mainstream computing labs, regular people who didn't have access to the mainframes and large computing data centres could use it.

C++'s impact in the digital world is immense. Many of the programs, applications, games, and even operating systems are coded using C++. For example, all of Adobe's major applications, such as Photoshop, InDesign and so on, are developed in C++. You will find that the browser you use to surf the Internet is written in C++, as well as Windows 10, Microsoft Office, and the backbone to Google's search engine. Apple's macOS is written largely in C++ (with some other languages mixed in depending on the function), and the likes of NASA, SpaceX, and even CERN use C++ for various applications, programs, controls, and umpteen other computing tasks.

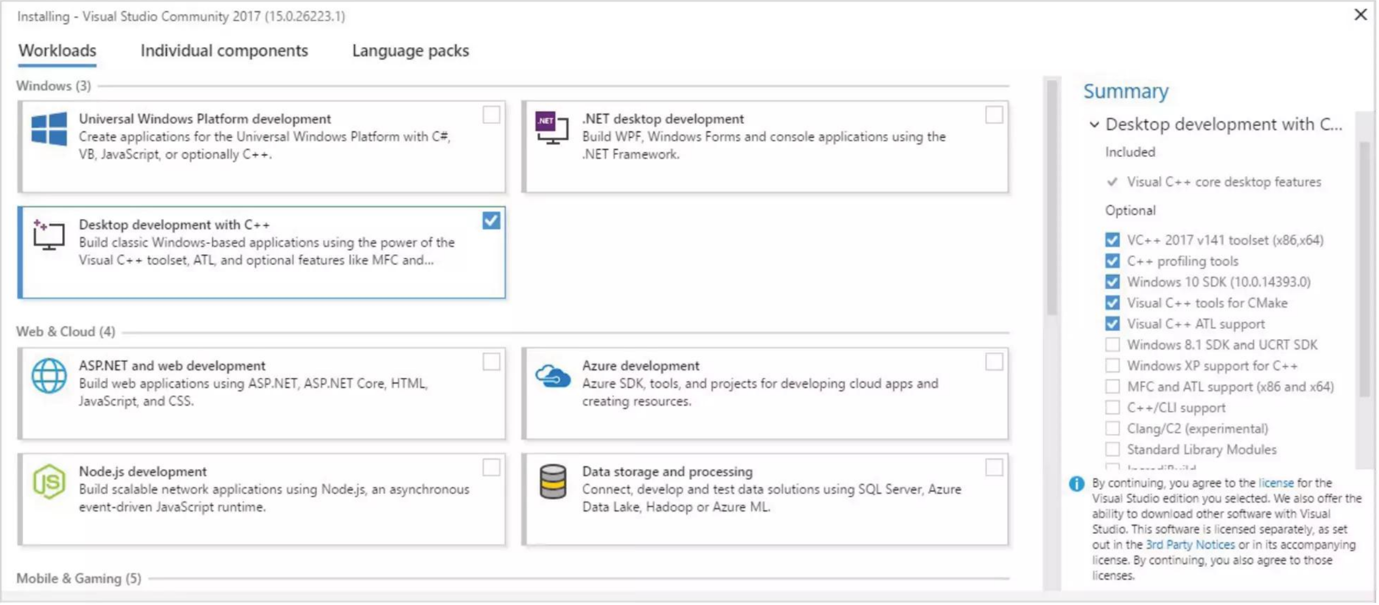
As well as being an easier addition to the core C language, C++ is also extremely efficient and performs well across the board. This higher level of performance over other languages, such as Python, BASIC and such, makes it an ideal development environment for modern computing; hence the aforementioned companies using it so widely.

C++ code is much faster than other programming languages.



```
#include <iostream>
using namespace std;
int main()
{
  int n, i;
  bool isPrime = true;
  cout << "Enter a positive integer: ";
  cin >> n;
  for(i = 2; i <= n / 2; ++i)
  {
      if(n % i == 0)
      {
          isPrime = false;
          break;
      }
  }
  if (isPrime)
      cout << "This is a prime number";
  else
      cout << "This is not a prime number";
  return 0;
}
```

*screenshot.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Start here
*screenshot.cpp
Logs & others
Build messages
Type here to search
ASUS



Windows (3)

☐ Universal Windows Platform development
Create applications for the Universal Windows Platform with C#, VB, JavaScript, or optionally C++.

☐ .NET desktop development
Build WPF, Windows Forms and console applications using the .NET Framework.

☑ Desktop development with C++
Build classic Windows-based applications using the power of the Visual C++ toolset, ATL, and optional features like MFC and...

Web & Cloud (4)

☐ ASP.NET and web development
Build web applications using ASP.NET, ASP.NET Core, HTML, JavaScript, and CSS.

☐ Azure development
Azure SDK, tools, and projects for developing cloud apps and creating resources.

☐ Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

☐ Data storage and processing
Connect, develop and test data solutions using SQL Server, Azure Data Lake, Hadoop or Azure ML.

Mobile & Gaming (5)

Desktop development with C...

Included

✓ Visual C++ core desktop features

Optional

☑ VC++ 2017 v141 toolset (x86,x64)
☑ C++ profiling tools
☑ Windows 10 SDK (10.0.14393.0)
☑ Visual C++ tools for CMake
☑ Visual C++ ATL support
☐ Windows 8.1 SDK and UCRT SDK
☐ Windows XP support for C++
☐ MFC and ATL support (x86 and x64)
☐ C++/CLI support
☐ Clang/C2 (experimental)
☐ Standard Library Modules

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Microsoft's Visual Studio is a great, free environment in which to learn C++.

C++ puts the developer in a much wider world of coding. By mastering C++, you will find yourself being able to develop code for the likes of Microsoft, Apple and so on. Generally, C++ developers enjoy a higher salary than programmers of some other languages, and, due to its versatility, the C++ programmer can move between jobs and companies without the need to re-learn anything specific.

You will discover, as you become a more advanced coder, that many of the developers in various coding jobs around the world tend to use pre-designed development engines. For example, when creating games, the likes of Bethesda, the team behind Oblivion and Skyrim, utilise a 3D game engine called The Creation Engine. This enables the team to quickly create animations, characters, items, terrains, rooms, and just about everything else you'd see in the game. The engine itself has been modified to make the most of the current graphics card hardware, and computer or console processing power. These engines are mostly written in C++, and when making improvements to the engine, or when creating a new game, if the developers want to add something that the engine can't do, they will use C++ to create the new content or link between two different engines. The end result, of course, is a game that contains the latest graphical technology, while being seamlessly bound together with some pretty clever C++ coding.

Getting to use C++ is quite easy, all you need is the right set of tools in which to communicate with the computer in C++, and you can start your journey. A C++ IDE is free of charge, even the immensely powerful Visual Studio from Microsoft is freely available to download and use. You can get into C++ from any operating system, be it macOS, Linux, Windows, or even mobile platforms.

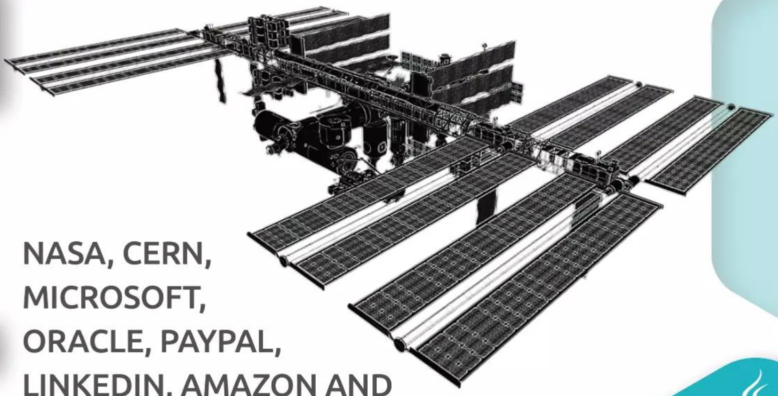
So, to answer the question of Why C++, the answer is because it's fast, efficient, and developed by most of the applications you regularly use. It's cutting edge, and a fantastic language to have mastered.

Indeed, the operating system you're using is written in C++.

Windows 10
debian
C++
Mac
ubuntu

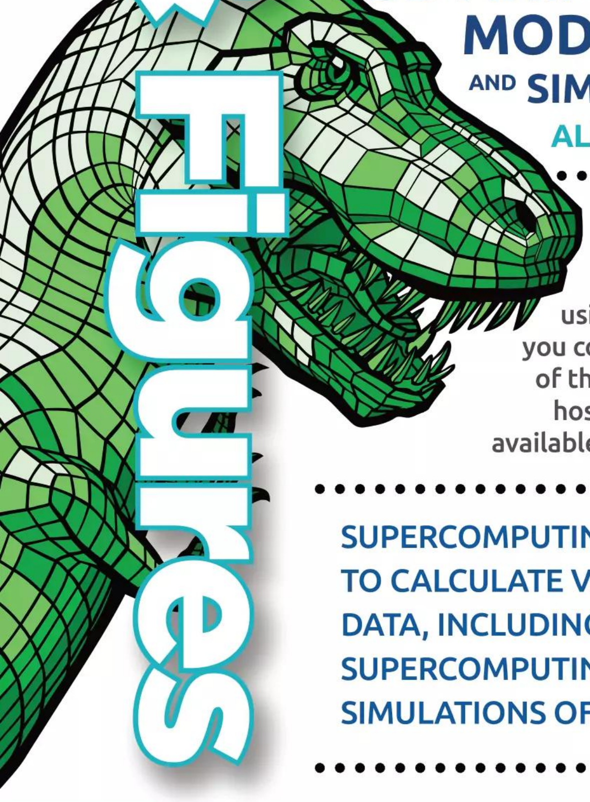
C++ Facts & Figures

C++ is one of the top programming languages in the industry. It's quick, powerful, and used by nearly every major tech and gaming company in the world. Here's some interesting facts about the rather wonderful C++.



NASA, CERN, MICROSOFT, ORACLE, PAYPAL, LINKEDIN, AMAZON AND THE MILITARY USE C++.

C++ is one of the predominant programming languages for the development of all kinds of technical and commercial software.



C++ IS USED FREQUENTLY IN ANIMATION PROCESSES. **3D ANIMATION, MODELLING,** AND **SIMULATIONS** ALL UTILISE C++.

C++ has influenced other programming languages, such as C# and Java.

A lot of the Linux operating system is coded using C++, therefore you could say that most of the world's Internet hosting servers are available thanks to C++.

70%

More than 70% of all trading is known as High Frequency Trading (HFT), and the software responsible is written in C++ to make use of its high speed.

SUPERCOMPUTING USES C++ CODE TO CALCULATE VAST AMOUNTS OF DATA, INCLUDING THE NASA-BASED SUPERCOMPUTING FARM THAT RUNS SIMULATIONS OF THE UNIVERSE.

Sources: ITJobsWatch, Google, Quora, Stack Exchange, eduCBA, StackOverflow, stackShare, AppDynamics

The Xbox operating system uses C++ as its backbone.

Many database applications are built using C++, such as MySQL, it's also used by Wikipedia, Yahoo and YouTube.

Gaming is one of the biggest users of C++. It handles the complexities of 3D games, supports multiplayer options, and enables intensive CPU and GPU hardware functions.

Google Chrome, Mozilla's Firefox, and even Microsoft's Edge web browsers are coded in C++.

C++ is used in computer networking as the main code behind the Programmable Logic Controller, connecting servers, processors, other hardware, and even robotics.

THE **90th** PERCENTILE SALARY FOR A UK-BASED C++ DEVELOPER IS **£120,000**

Most of Adobe's stable of products are developed using C++.

Windows 95, 98, 2000, XP, 7, 8.1 and 10, as well as Microsoft Office, use C++ as the backbone programming language of choice.

MRI scanning machines and Computer Aided Design all use C++ to help with the enhanced imagery produced by these systems.

The Unreal 4 Engine, which is coded in C++, is used for hundreds of games, including Fortnite.

RADAR PROCESSING TECHNOLOGIES USE C++, AS WELL AS ADVANCED MISSILE DETECTION SYSTEMS.

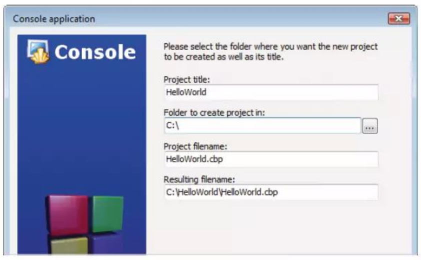
FLIGHT CONTROL SYSTEMS IN MODERN AIRCRAFT, INCLUDING MILITARY AIRCRAFT, USE C++.

Equipment You Will Need

You don't need to invest a huge amount of money in order to learn C++, and you don't need an entire computing lab at your disposal either. Providing you have a fairly modern computer, everything else is freely available.

C++ SETUPS

As most, if not all, operating systems have C++ at their code, it stands to reason that you can learn to program in C++ no matter what OS you're currently using.



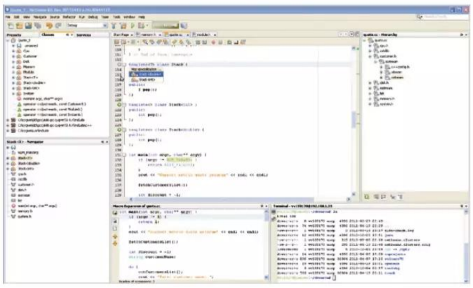
Please select the folder where you want the new project to be created as well as its title.

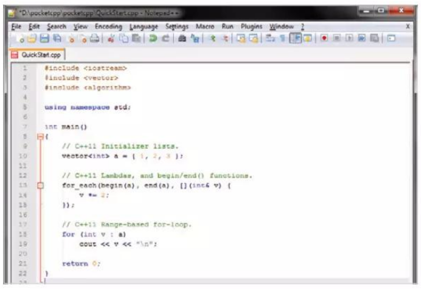
Project title: HelloWorld

Folder to create project in: C:\

Project filename: HelloWorld.cbp

Resulting filename: C:\HelloWorld\HelloWorld.cbp





```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
    // C++11 Initializer lists.
    vector<int> a = { 1, 2, 3 };

    // C++11 Lambdas, and begin/end() functions.
    for_each(begin(a), end(a), [](int& v) {
        v *= 2;
    });

    // C++11 Range-based for-loop.
    for (int v : a)
        cout << v << "\n";

    return 0;
}
```

☐ COMPUTER

Unless you fancy writing out your C++ code by hand on a sheet of paper (which is something many older coders used to do), then a computer is an absolute must have component. PC users can have any recent Linux distro or Windows OS, Mac users the latest macOS.

☐ AN IDE

An IDE is used to enter and execute your C++ code. Many IDEs come with extensions and plugins that help make it work better, or add an extra level of functionality. Often, an IDE will provide enhancements depending on the core OS being used, such as enhancements for Windows 10.

☐ COMPILER

A compiler is a program that converts the C++ language into binary that the computer can understand. While some IDEs come with a compiler built in, others don't. Code::Blocks is our favourite IDE that comes with a C++ compiler as part of the package. More on this later.

☐ TEXT EDITOR

Some programmers much prefer to use a text editor to assemble their C++ code before running it through a compiler. Essentially, you can use any text editor to write code, just save it with a .cpp extension. However, Notepad++ is one of the best code text editors available.

☐ INTERNET ACCESS

While it's entirely possible to learn how to code on a computer that's not attached to the Internet, it's extraordinarily difficult. You will need to install the relevant software, keep it up to date, install any extras or extensions, and look for help when coding. All of which require access to the Internet.

☐ TIME AND PATIENCE

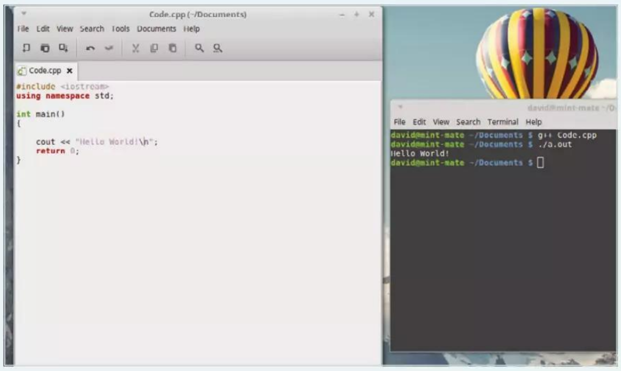
You're going to need to set aside significant time to spend on learning how to code in C++. Sadly, unless you're a genius, it's not going to happen overnight, or even in a week. A good C++ coder has spent many years honing their craft, so be patient, start small and keep learning.

OS SPECIFIC NEEDS

C++ will work in any operating system; however, getting all the necessary pieces together can be confusing to a newcomer. Here's some OS specifics for C++.

LINUX

Linux users are lucky in that they already have a compiler and text editor built into their operating system. Any text editor will allow you to type out your C++ code, when it's saved with a .cpp extension, use g++ to compile it.

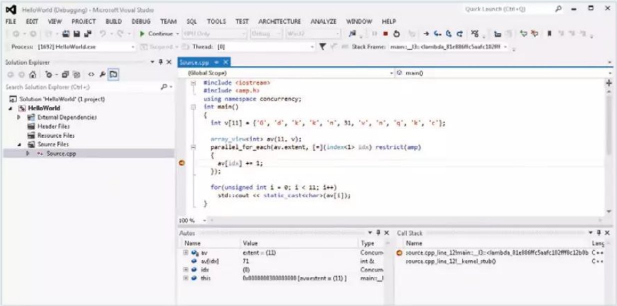


```
#include <iostream>
using namespace std;

int main()
{

    cout << "Hello World!\n";
    return 0;
}
```

```
david@mint-mate ~/Documents $ g++ Code.cpp
david@mint-mate ~/Documents $ ./a.out
Hello World!
david@mint-mate ~/Documents $ 
```

WINDOWS

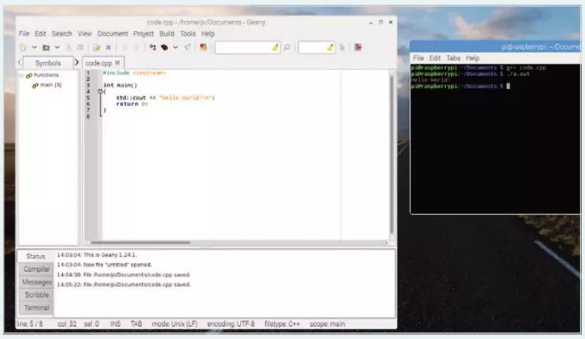
As we've mentioned previously, one good IDE is Microsoft's Visual Studio. However, a better IDE and compiler is Code::Blocks, which is kept regularly up to date with a new release twice yearly, or so. Otherwise, Windows users can enter their code in Notepad++ then compile it with MinGW – which Code::Blocks uses.



```
#include <iostream>
#include <amp.h>
using namespace concurrency;
int main()
{
  int v[11] = {'G', 'd', 'k', 'k', 'n', 31, 'v', 'n', 'q', 'k', 'c'};

  array_view<int> av(11, v);
  parallel_for_each(av.extent, [=](index<1> idx) restrict(amp)
  {
    av[idx] += 1;
  });

  for(unsigned int i = 0; i < 11; i++)
    std::cout << static_cast<char>(av[i]);
}
```

RASPBERRY PI

The Raspberry Pi's operating system is Raspbian, which is Linux based. Therefore, you're able to write your code out using a text editor and then compile it with g++, as you would in any other Linux distro.



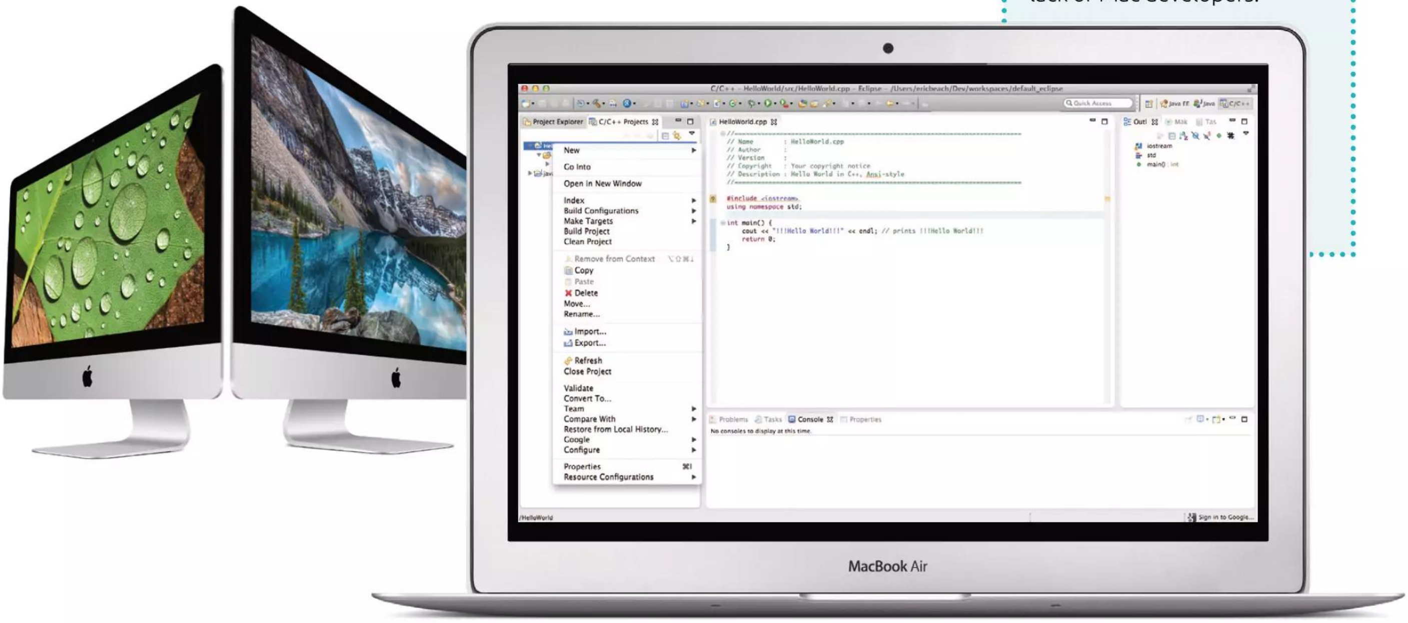
```
#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
    return 0;
}
```

```
14:03:04: This is Geany 1.24.1.
14:03:04: New file "untitled" opened.
14:04:38: File /home/pi/Documents/code.cpp saved.
14:05:22: File /home/pi/Documents/code.cpp saved.
```

```
pi@raspberrypi:~/Documents $ g++ code.cpp
pi@raspberrypi:~/Documents $ ./a.out
Hello World!
pi@raspberrypi:~/Documents $ 
```

MAC

Mac owners will need to download and install Xcode, in order to compile their C++ code natively. Other options for the macOS include Netbeans, Eclipse or Code::Blocks. Note: the latest Code::Blocks isn't available for Mac, due to a lack of Mac developers.



```
//============================================================================
// Name        : HelloWorld.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
using namespace std;

int main() {
	cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
	return 0;
}
```

Structure of C++ Code

C++ is an amazing programming language to learn. If your dream is to become a games designer, or work at the cutting edge of science or engineering technology, then being able to code in C++ is a must. Remember, you're never too old to learn how to code.

#INCLUDE <C++ IS ACE!>

As you learn the basics of programming, you will begin to understand the structure of a program. The commands may be different, but you will start to see how the code works.

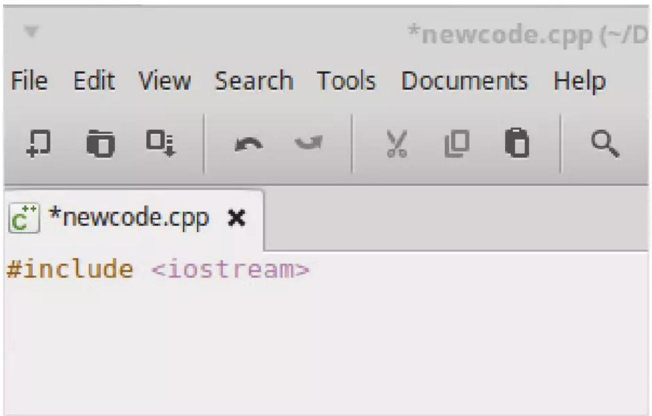
C++

Danish student Bjarne Stroustrup invented C++ in 1979, as a part of his Ph.D. thesis. Initially C++ was called C with Classes, which added features to the already popular C programming language while making it a more user-friendly environment.

Bjarne Stroustrup, inventor of C++.

#INCLUDE

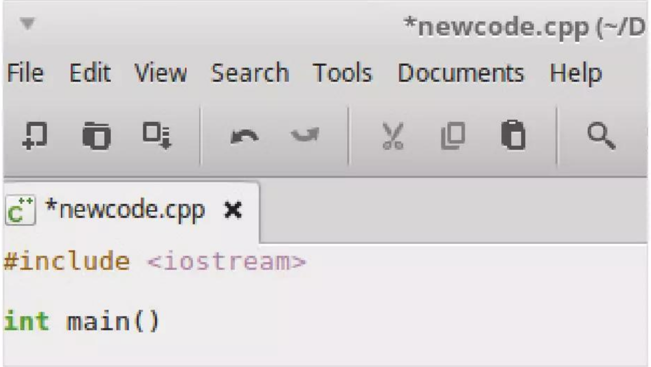
The structure of a C++ program can look complex, but once you get familiar with it you'll begin to see how it flows. Every C++ code begins with a directive, #include <>. The directive instructs the pre-processor to include a section of the standard C++ code. For example: `#include <iostream>` includes the iostream header to support input/output operations.


*newcode.cpp (~/D
File Edit View Search Tools Documents Help
*newcode.cpp

```
#include <iostream>
```

INT MAIN()

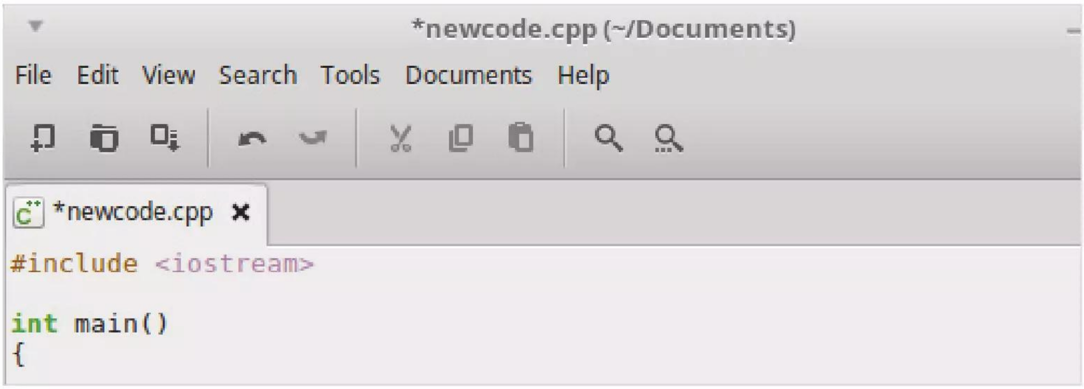
int main() initiates the declaration of a function, which is a group of code statements under the name 'main'. All C++ code begins at the main function, regardless of where it lies within the main body of the code.


*newcode.cpp (~/D
File Edit View Search Tools Documents Help
*newcode.cpp

```
#include <iostream>

int main()
```

BRACES

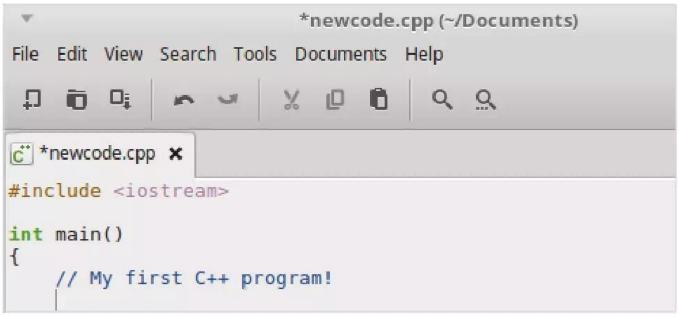
The open brace is something that you may not have come across before, especially if you're used to other coding languages. The open brace indicates the beginning of the main function, and contains all the code belonging to that function.


*newcode.cpp (~/Documents)
File Edit View Search Tools Documents Help
*newcode.cpp

```
#include <iostream>

int main()
{
```

COMMENTS

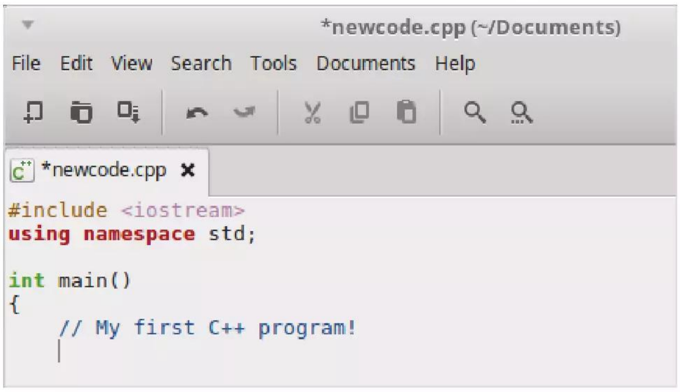
Lines that begin with a double slash are comments. This means they won't be executed in the code and are ignored by the compiler. Why are they there? Comments are designed to help you, or another programmer looking at your code, explain what's going on. There are two types of comment: /* covers multiple line comments, // a single line.


*newcode.cpp (~/Documents)
File Edit View Search Tools Documents Help
*newcode.cpp

```
#include <iostream>

int main()
{
    // My first C++ program!
```

STD

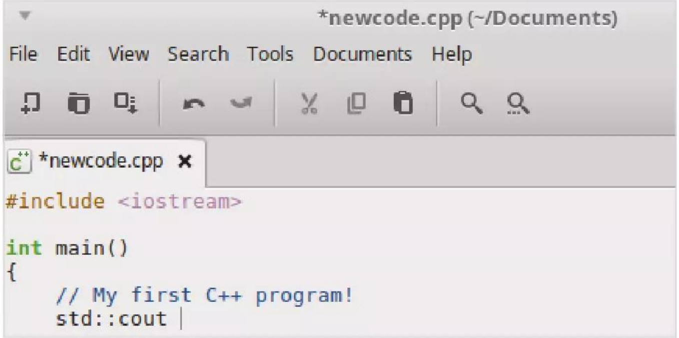
In C++, STD means Standard. It's a part of the Standard Namespace in C++, which covers a number of different statements and commands. You can leave the std part out of a code, but it must be declared at the start with: `using namespace std`.


*newcode.cpp (~/Documents)
File Edit View Search Tools Documents Help
*newcode.cpp

```
#include <iostream>
using namespace std;

int main()
{
    // My first C++ program!
```

COUT

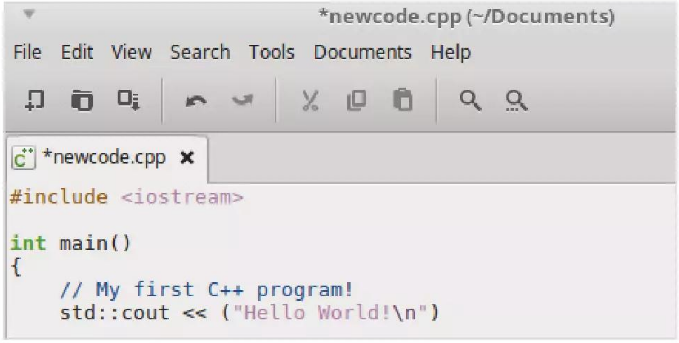
In this example we're using cout, which is a part of the Standard Namespace – hence why it's there, as you're asking C++ to use it from that particular namespace. Cout means Character OUTput, which displays, or prints, something to the screen. If we leave `std::` out we have to declare it at the start of the code; as mentioned previously.


*newcode.cpp (~/Documents)
File Edit View Search Tools Documents Help
*newcode.cpp

```
#include <iostream>

int main()
{
    // My first C++ program!
    std::cout
```

<<

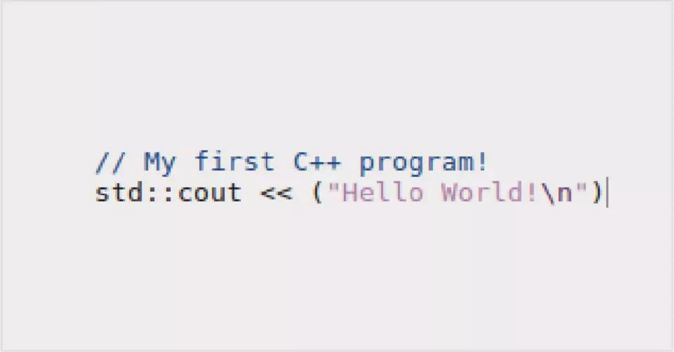
The two chevrons used here are insertion operators. This means that, whatever follows, the chevrons are to be inserted into the std::cout statement. In this case, they are the words 'Hello World', which are to be displayed on the screen when you compile and execute the code.


*newcode.cpp (~/Documents)
File Edit View Search Tools Documents Help
*newcode.cpp

```
#include <iostream>

int main()
{
    // My first C++ program!
    std::cout << ("Hello World!\n")
```

OUTPUTS

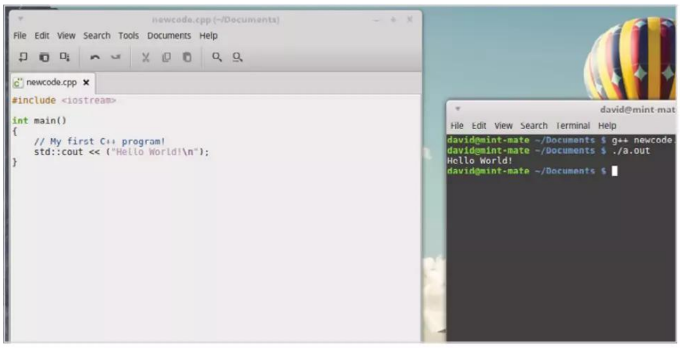
Leading on, ("Hello World!") is the part that we want to appear on the screen when the code is executed. You can enter whatever you like, as long as it's inside the quotation marks. The brackets aren't needed, but some compilers insist on them. The \n part indicates a new line is to be inserted.



```
// My first C++ program!
std::cout << ("Hello World!\n")
```

; AND }

Finally you will notice that lines within a function code block (except comments) end with a semicolon. This marks the end of the statement, and all statements in C++ must have one at the end or the compiler will fail to build the code. The very last line has the closing brace to indicate the end of the main function.


newcode.cpp (~/Documents)
File Edit View Search Tools Documents Help
newcode.cpp
david@mint-mat
File Edit View Search Terminal Help

```
#include <iostream>

int main()
{
    // My first C++ program!
    std::cout << ("Hello World!\n");
}
```

```
david@mint-mate ~/Documents $ g++ newcode
david@mint-mate ~/Documents $ ./a.out
Hello World!
david@mint-mate ~/Documents $ 
```

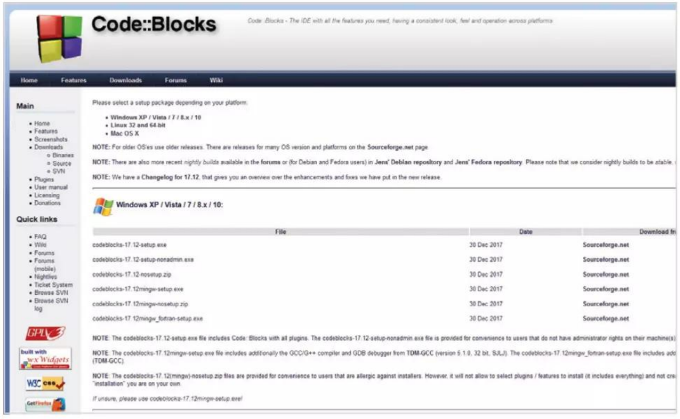
How to Set Up C++ in Windows

Windows users have a wealth of choice when it comes to programming in C++. There are loads of IDEs and compilers available, including Visual Studio from Microsoft. However, in our opinion, the best C++ IDE to begin with is Code::Blocks.

CODE::BLOCKS

Code::Blocks is a free C++, C and Fortran IDE that is feature rich and easily extendible with plugins. It's easy to use, comes with a compiler and has a vibrant community behind it too.

STEP 1 Start by visiting the Code::Blocks download site, at **www.codeblocks.org/downloads**. From there, click on the 'Download the binary releases' link to be taken to the latest downloadable version for Windows.



Please select a setup package depending on your platform:

- Windows XP / Vista / 7 / 8.x / 10
- Linux 32 and 64-bit
- Mac OS X

NOTE: For older OS'es use older releases. There are releases for many OS version and platforms on the **Sourceforge.net** page

NOTE: There are also more recent *nightly builds* available in the **forums** or (for Debian and Fedora users) in **Jens' Debian repository** and **Jens' Fedora repository**. Please note that we consider nightly builds to be *stable*,

NOTE: We have a **Changelog for 17.12**, that gives you an overview over the enhancements and fixes we have put in the new release.

Windows XP / Vista / 7 / 8.x / 10:

File	Date	Download fr
codeblocks-17.12-setup.exe	30 Dec 2017	Sourceforge.net
codeblocks-17.12-setup-nonadmin.exe	30 Dec 2017	Sourceforge.net
codeblocks-17.12-nosetup.zip	30 Dec 2017	Sourceforge.net
codeblocks-17.12mingw-setup.exe	30 Dec 2017	Sourceforge.net
codeblocks-17.12mingw-nosetup.zip	30 Dec 2017	Sourceforge.net
codeblocks-17.12mingw_fortran-setup.exe	30 Dec 2017	Sourceforge.net

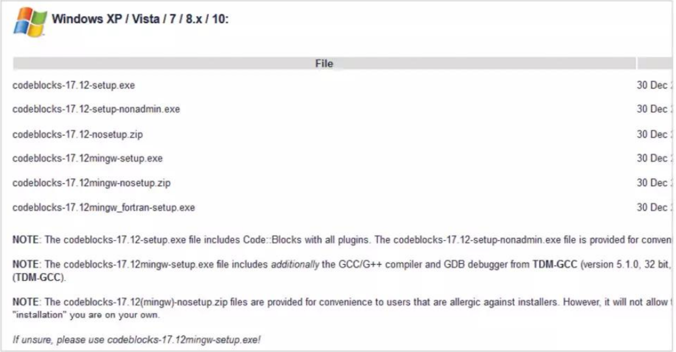
NOTE: The codeblocks-17.12-setup.exe file includes Code::Blocks with all plugins. The codeblocks-17.12-setup-nonadmin.exe file is provided for convenience to users that do not have administrator rights on their machine(s)

NOTE: The codeblocks-17.12mingw-setup.exe file includes additionally the GCC/G++ compiler and GDB debugger from TDM-GCC (version 5.1.0, 32 bit, SJLJ). The codeblocks-17.12mingw_fortran-setup.exe file includes add (TDM-GCC)

NOTE: The codeblocks-17.12(mingw)-nosetup.zip files are provided for convenience to users that are allergic against installers. However, it will not allow to select plugins / features to install (it includes everything) and not cre "installation" you are on your own.

If unsure, please use codeblocks-17.12mingw-setup.exe!

STEP 2 There you can see, there are several Windows versions available. The one you want to download has mingw-setup.exe at the end of the current version number. At the time of writing this is: codeblocks-17.12mingw-setup.exe. The difference is that the mingw-setup version includes a C++ compiler and debugger from TDM-GCC (a compiler suite).



Windows XP / Vista / 7 / 8.x / 10:

File	
codeblocks-17.12-setup.exe	30 Dec
codeblocks-17.12-setup-nonadmin.exe	30 Dec
codeblocks-17.12-nosetup.zip	30 Dec
codeblocks-17.12mingw-setup.exe	30 Dec
codeblocks-17.12mingw-nosetup.zip	30 Dec
codeblocks-17.12mingw_fortran-setup.exe	30 Dec

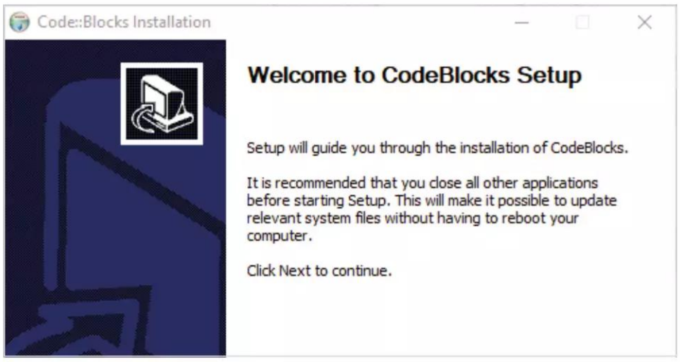
NOTE: The codeblocks-17.12-setup.exe file includes Code::Blocks with all plugins. The codeblocks-17.12-setup-nonadmin.exe file is provided for conven

NOTE: The codeblocks-17.12mingw-setup.exe file includes *additionally* the GCC/G++ compiler and GDB debugger from **TDM-GCC** (version 5.1.0, 32 bit, (TDM-GCC).

NOTE: The codeblocks-17.12(mingw)-nosetup.zip files are provided for convenience to users that are allergic against installers. However, it will not allow "installation" you are on your own.

If unsure, please use codeblocks-17.12mingw-setup.exe!

STEP 3 When you've located the file, click on the Sourceforge.net link at the end of the line and a download notification window appears; click on Save File to start the download and save the executable to your PC. Locate the downloaded Code::Blocks installer and double-click to start. Follow the on-screen instructions to begin the installation.



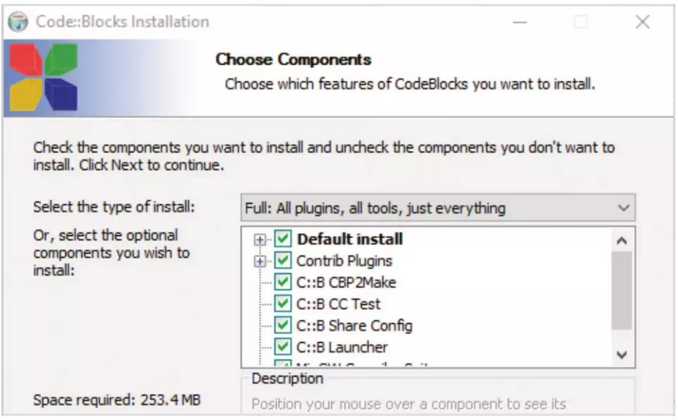
Welcome to CodeBlocks Setup

Setup will guide you through the installation of CodeBlocks.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

STEP 4 Once you agree to the licencing terms, a choice of installation options becomes available. You can opt for a smaller install, missing out on some of the components but we recommend that you opt for the Full option, as default.



Choose Components

Choose which features of CodeBlocks you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Full: All plugins, all tools, just everything

Or, select the optional components you wish to install:

- ☑ **Default install**
- ☑ Contrib Plugins
- ☑ C::B CBP2Make
- ☑ C::B CC Test
- ☑ C::B Share Config
- ☑ C::B Launcher

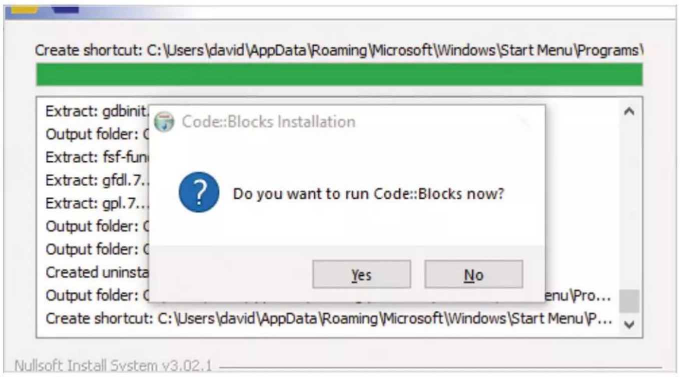
Space required: 253.4 MB

Description

Position your mouse over a component to see its

STEP 5

Next choose an install location for the Code::Blocks files. It's your choice but the default is generally sufficient (unless you have any special requirements of course). When you click Next, the install begins; when it's finished a notification pops up asking you if you want start Code::Blocks now, so click Yes.



```
Create shortcut: C:\Users\david\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\
Extract: gdbinit
Output folder: C
Extract: fsf-fun
Extract: gfdl.7..
Extract: gpl.7...
Output folder: C
Output folder: C
Created uninsta
Output folder: C                                        enu\Pro...
Create shortcut: C:\Users\david\AppData\Roaming\Microsoft\Windows\Start Menu\P...
```

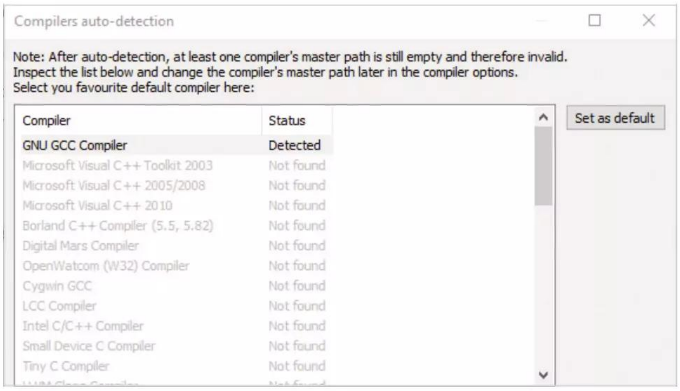
Code::Blocks Installation

Do you want to run Code::Blocks now?

Yes
No
Nullsoft Install System v3.02.1

STEP 6

The first time Code::Blocks loads it runs an autodetect for any C++ compilers you may already have installed on your system. If you don't have any, click on the first detected option: GNU GCC Compiler and click the Default button to set it as the system's C++ compiler. Click OK when you're ready to continue.



Compilers auto-detection

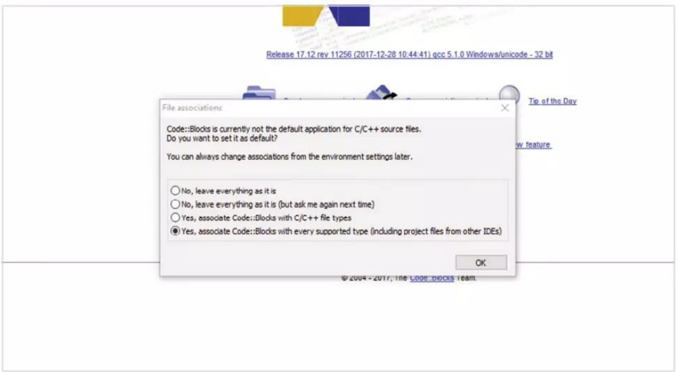
Note: After auto-detection, at least one compiler's master path is still empty and therefore invalid.
Inspect the list below and change the compiler's master path later in the compiler options.
Select you favourite default compiler here:

Compiler	Status
GNU GCC Compiler	Detected
Microsoft Visual C++ Toolkit 2003	Not found
Microsoft Visual C++ 2005/2008	Not found
Microsoft Visual C++ 2010	Not found
Borland C++ Compiler (5.5, 5.82)	Not found
Digital Mars Compiler	Not found
OpenWatcom (W32) Compiler	Not found
Cygwin GCC	Not found
LCC Compiler	Not found
Intel C/C++ Compiler	Not found
Small Device C Compiler	Not found
Tiny C Compiler	Not found

Set as default

STEP 7

The program starts and another message appears informing you that Code::Blocks is currently not the default application for C++ files. You have two options, to leave everything as it is or allow Code::Blocks to associate all C++ file types. Again, we would recommend you opt for the last choice, to associate Code::Blocks with every supported file type.



Release 17.12 rev 11256 (2017-12-28 10:44:41) gcc 5.1.0 Windows/unicode - 32 bit

File associations

Code::Blocks is currently not the default application for C/C++ source files.
Do you want to set it as default?

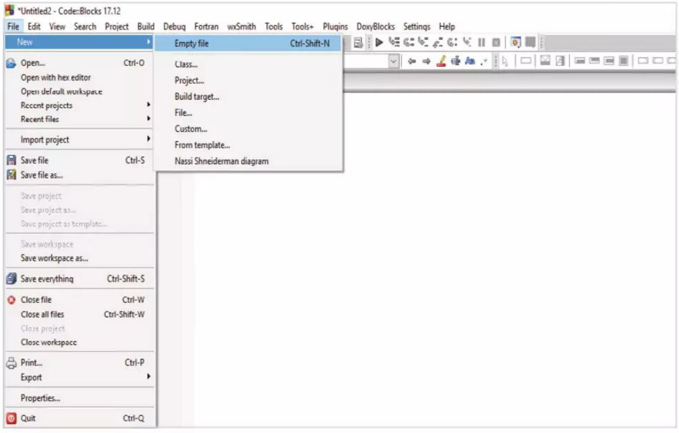
You can always change associations from the environment settings later.

○ No, leave everything as it is
○ No, leave everything as it is (but ask me again next time)
○ Yes, associate Code::Blocks with C/C++ file types
● Yes, associate Code::Blocks with every supported type (including project files from other IDEs)

OK
Tip of the Day
w feature

STEP 8

There's a lot you can do in Code::Blocks, so you need to dig in and find a good C++ tutorial to help you get the most from it. However, to begin with, click on File > New > Empty File. This creates a new, blank window for you to type in.



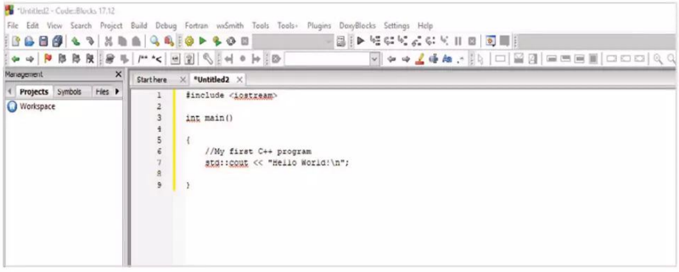
*Untitled2 - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
New
Open... Ctrl-O
Open with hex editor
Open default workspace
Recent projects
Recent files
Import project
Save file Ctrl-S
Save file as...
Save project
Save project as...
Save project as template...
Save workspace
Save workspace as...
Save everything Ctrl-Shift-S
Close file Ctrl-W
Close all files Ctrl-Shift-W
Close project
Close workspace
Print... Ctrl-P
Export
Properties...
Quit Ctrl-Q
Empty file Ctrl-Shift-N
Class...
Project...
Build target...
File...
Custom...
From template...
Nassi Shneiderman diagram

STEP 9

In the new window, enter the following:

```
#include <iostream>

Int main()
{
//My first C++ program
Std::cout << “Hello World!\n”;
}
```

Notice how Code::Blocks auto-inserts the braces and speech quotes.

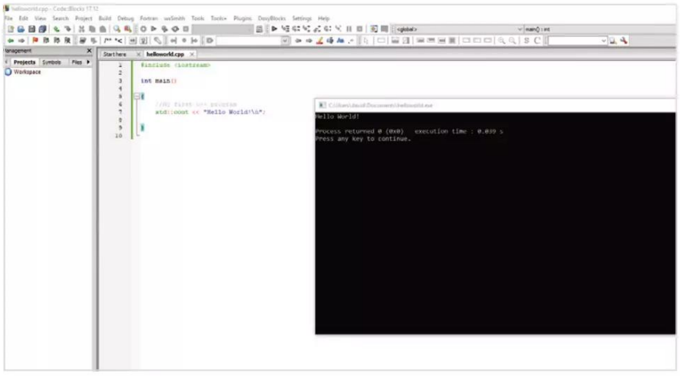


```
#include <iostream>

int main()

{
    //My first C++ program
    std::cout << "Hello World!\n";

}
```

*Untitled2 - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Symbols Files
Workspace
Start here
*Untitled2

STEP 10

Click File > Save as and save the code with a .cpp extension (helloworld.cpp, for example). Code::Blocks changes the view to colour code according to C++ standards. To execute the code, click on the Build and Run icon along the top of the screen. It's a green play icon together with a yellow cog.



```
Hello World!

Process returned 0 (0x0)   execution time : 0.039 s
Press any key to continue.
```

helloworld.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
main() : int
Management
Projects Symbols Files
Workspace
Start here
helloworld.cpp

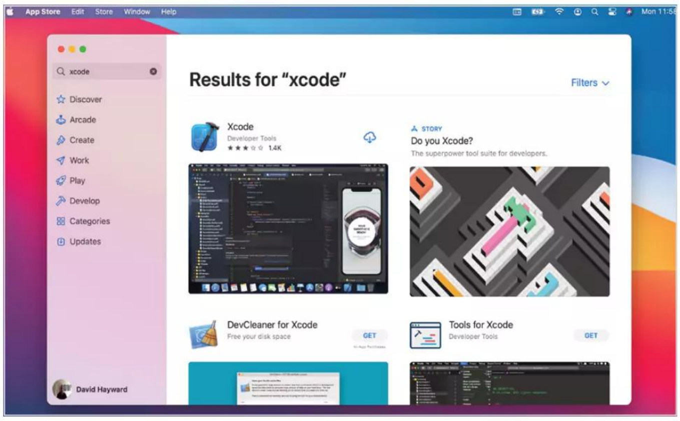
How to Set Up C++ on a Mac

To begin C++ coding on a Mac you can use Apple's own developer platform: Xcode. This is a free, full featured IDE that's designed to create native Apple apps. However, it's also able to be used to create C++ code relatively easily.

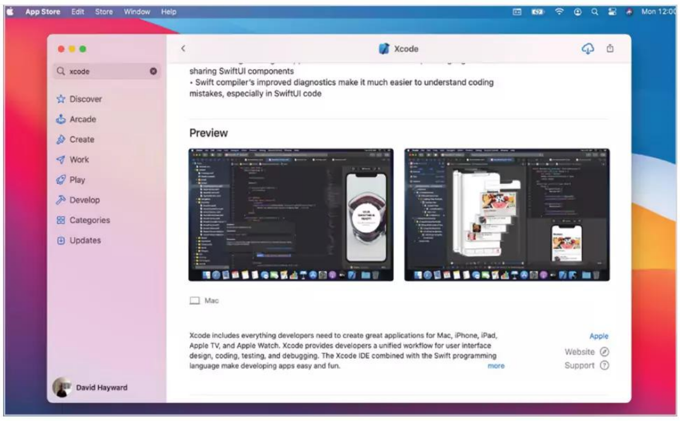
XCODE

Apple's Xcode is primarily designed for users to develop apps for macOS, iOS, tvOS and watchOS applications in Swift or Objective-C, but we can use it for C++ too.

STEP 1 Start by opening the App Store on your Mac, Apple Menu > App Store. In the Search box enter Xcode, and press Return. There will be many suggestions filling the App Store window, but it's first option, Xcode, that you need to click on.


App Store
Edit
Store
Window
Help
xcode
Discover
Arcade
Create
Work
Play
Develop
Categories
Updates
David Hayward
Results for "xcode"
Filters
Xcode
Developer Tools
1.4K
STORY
Do you Xcode?
The superpower tool suite for developers.
DevCleaner for Xcode
Free your disk space
GET
Tools for Xcode
Developer Tools
GET

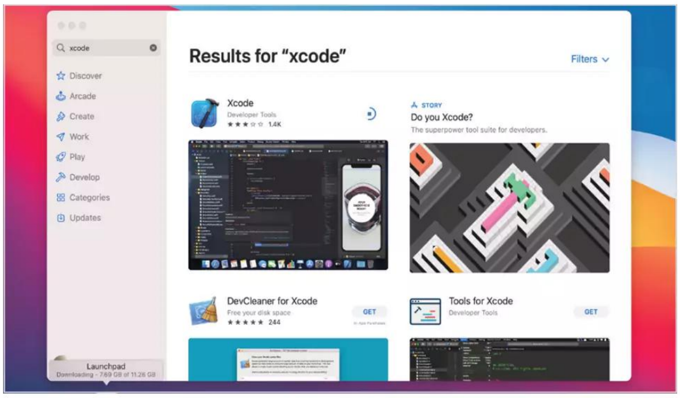
STEP 2 Take a moment to browse through the app's information, including the compatibility to ensure you have the correct version of macOS. Xcode requires macOS 10.12.6 or later to install and work.


App Store
Edit
Store
Window
Help
xcode
Discover
Arcade
Create
Work
Play
Develop
Categories
Updates
David Hayward
Xcode
Preview
Mac
Apple
Website
Support
more

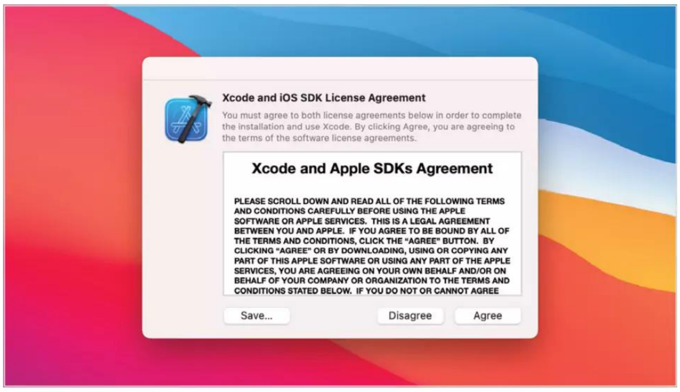
sharing SwiftUI components
• Swift compiler's improved diagnostics make it much easier to understand coding mistakes, especially in SwiftUI code

Xcode includes everything developers need to create great applications for Mac, iPhone, iPad, Apple TV, and Apple Watch. Xcode provides developers a unified workflow for user interface design, coding, testing, and debugging. The Xcode IDE combined with the Swift programming language make developing apps easy and fun.

STEP 3 When you're ready, click on the Get or cloud icon button which will install the Xcode app. Enter your Apple ID, and Xcode will begin to download and install. It may take some time depending on the speed of your Internet connection, as Xcode is in excess of 11GB.


xcode
Discover
Arcade
Create
Work
Play
Develop
Categories
Updates
Results for "xcode"
Filters
Xcode
Developer Tools
1.4K
STORY
Do you Xcode?
The superpower tool suite for developers.
DevCleaner for Xcode
Free your disk space
244
GET
Tools for Xcode
Developer Tools
GET
Launchpad
Downloading - 7.69 GB of 11.28 GB

STEP 4 When the installation is complete, click on the Open button to launch Xcode. Click Agree to the licence terms, and enter your password to allow Xcode to make changes to the system. When you've done that, Xcode will begin to install additional components.


Save...
Disagree
Agree

Xcode and iOS SDK License Agreement

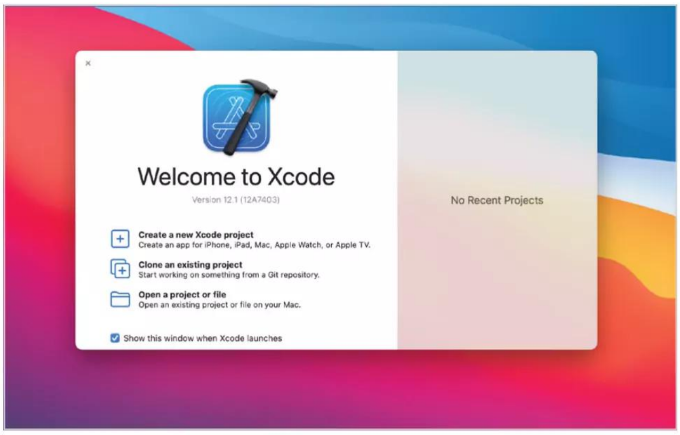
You must agree to both license agreements below in order to complete the installation and use Xcode. By clicking Agree, you are agreeing to the terms of the software license agreements.

Xcode and Apple SDKs Agreement

PLEASE SCROLL DOWN AND READ ALL OF THE FOLLOWING TERMS AND CONDITIONS CAREFULLY BEFORE USING THE APPLE SOFTWARE OR APPLE SERVICES. THIS IS A LEGAL AGREEMENT BETWEEN YOU AND APPLE. IF YOU AGREE TO BE BOUND BY ALL OF THE TERMS AND CONDITIONS, CLICK THE "AGREE" BUTTON. BY CLICKING "AGREE" OR BY DOWNLOADING, USING OR COPYING ANY PART OF THIS APPLE SOFTWARE OR USING ANY PART OF THE APPLE SERVICES, YOU ARE AGREEING ON YOUR OWN BEHALF AND/OR ON BEHALF OF YOUR COMPANY OR ORGANIZATION TO THE TERMS AND CONDITIONS STATED BELOW. IF YOU DO NOT OR CANNOT AGREE

STEP 5

With everything now installed, including the additional components, Xcode will launch displaying the version number along with three choices and any recent projects that you've worked on – although for a fresh install, this will be blank.



Welcome to Xcode

Version 12.1 (12A7403)

Create a new Xcode project
Create an app for iPhone, iPad, Mac, Apple Watch, or Apple TV.

Clone an existing project
Start working on something from a Git repository.

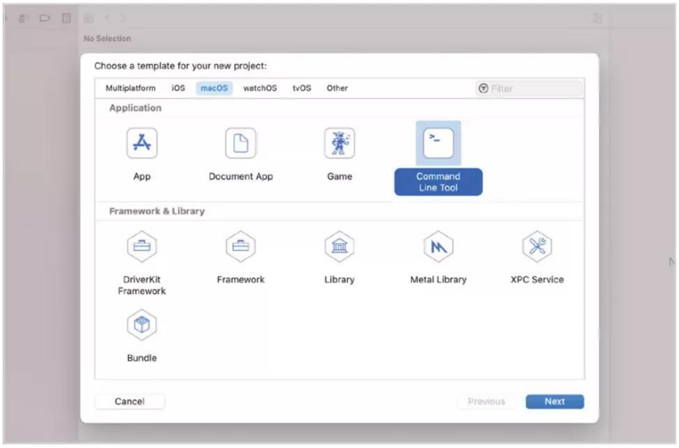
Open a project or file
Open an existing project or file on your Mac.

☑ Show this window when Xcode launches

No Recent Projects

STEP 6

Start by clicking on Create New Xcode Project, this opens a template window to choose which platform you're developing code for. Click the macOS tab, then click the Command Line Tool option. Click Next to continue.

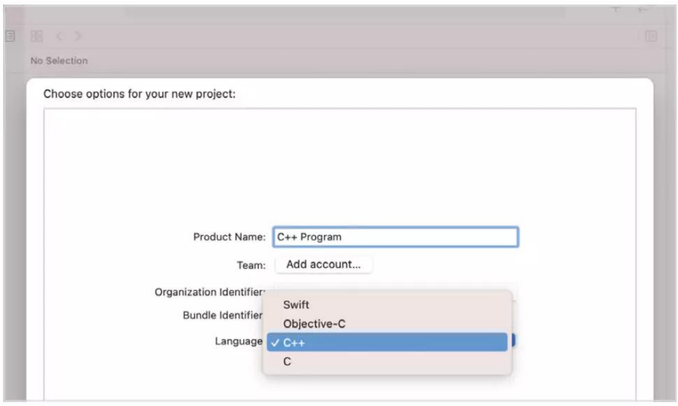


Choose a template for your new project:

Multiplatform
iOS
macOS
watchOS
tvOS
Other
Filter
Application
App
Document App
Game
Command Line Tool
Framework & Library
DriverKit Framework
Framework
Library
Metal Library
XPC Service
Bundle
Cancel
Previous
Next

STEP 7

Fill in all the fields, but ensure that the Language option at the bottom is set to C++. Simply choose it from the drop-down list. When you've filled in the fields, and made sure that C++ is the chosen language, click on the Next button to continue.



Choose options for your new project:

Product Name: C++ Program

Team: Add account...

Organization Identifier

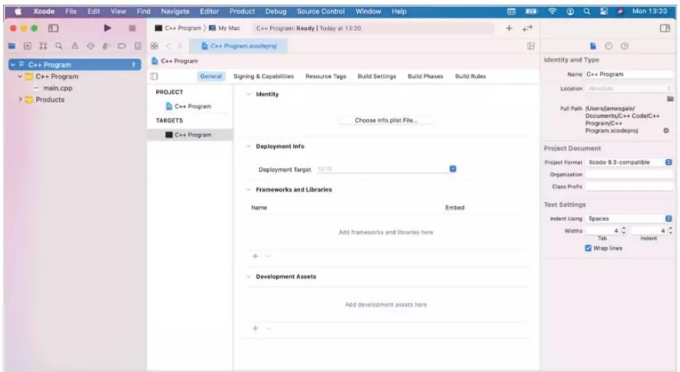
Bundle Identifier

Language: ✓ C++

Swift
Objective-C
C++
C

STEP 8

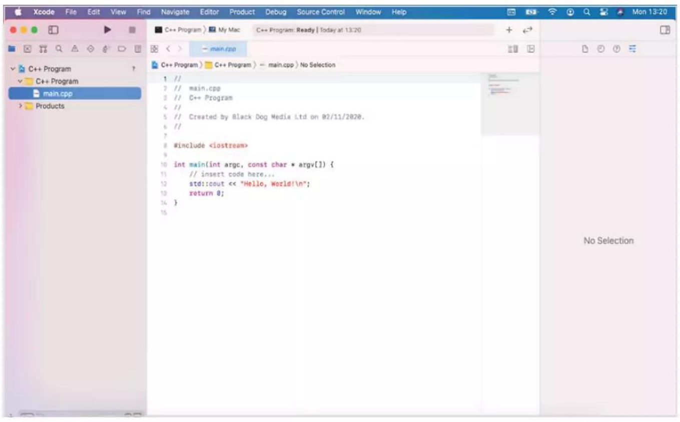
The next step asks where to create a Git Repository for all your future code. Choose a location on your Mac, or a network location, and click the Create button. When you've done all that, you can start to code. The left-hand pane details the files used in the C++ program you're coding. Click on the main.cpp file in the list.



Xcode
File
Edit
View
Find
Navigate
Editor
Product
Debug
Source Control
Window
Help
C++ Program
main.cpp
Products
General
Signing & Capabilities
Resource Tags
Build Settings
Build Phases
Build Rules
PROJECT
TARGETS
Identity
Deployment Info
Frameworks and Libraries
Development Assets
Identity and Type
Project Document
Text Settings

STEP 9

You will notice that Xcode has automatically completed a basic Hello World program for you. The differences here are that the int main () function now contains multiple functions and the layout is slightly different. This is just Xcode utilising the content that's available to your Mac.



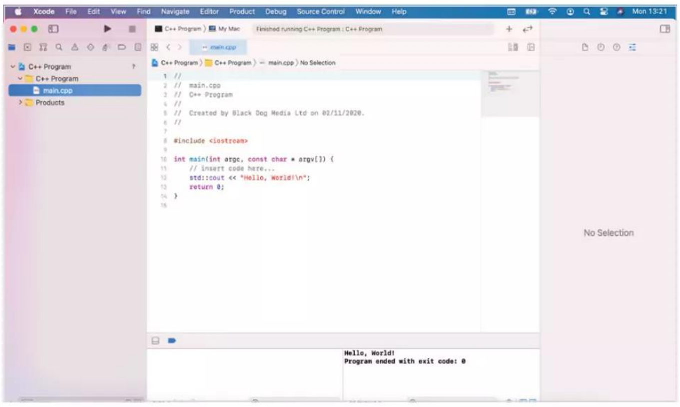
C++ Program: Ready | Today at 13:20

```
//
//  main.cpp
//  C++ Program
//
//  Created by Black Dog Media Ltd on 02/11/2020.
//

#include <iostream>

int main(int argc, const char * argv[]) {
    // insert code here...
    std::cout << "Hello, World!\n";
    return 0;
}
```

No Selection

STEP 10

When you want to run the code, click on Product > Run. You may be asked to enable Developer Mode on the Mac, this is to authorise Xcode to perform functions without needing your password every session. When the program executes, the output will be displayed at the bottom of the Xcode window.



Finished running C++ Program : C++ Program

```
//
//  main.cpp
//  C++ Program
//
//  Created by Black Dog Media Ltd on 02/11/2020.
//

#include <iostream>

int main(int argc, const char * argv[]) {
    // insert code here...
    std::cout << "Hello, World!\n";
    return 0;
}
```

```
Hello, World!
Program ended with exit code: 0
```

No Selection

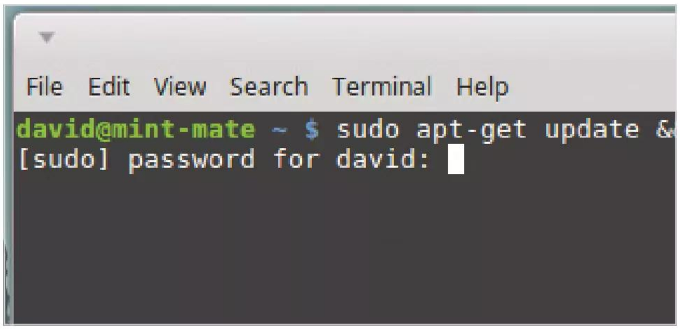
How to Set Up C++ in Linux

Linux is a great C++ coding environment. Most Linux distros already have the essential components preinstalled, such as a compiler and the text editors are excellent for entering code into, including colour coding; there's also tons of extra software available to help you out.

LINUX++

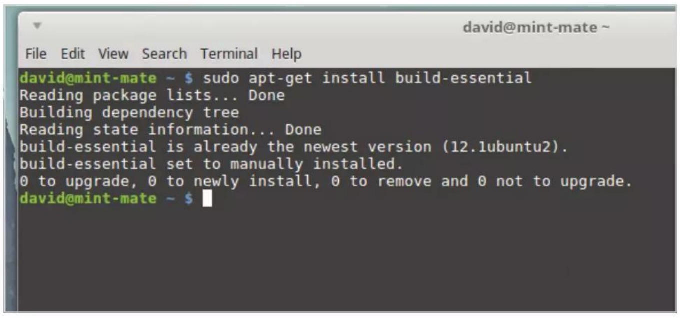
There are many different versions of Linux available, for this example we're using one of the more popular distributions: Linux Mint. However, these steps will work in any Debian-based Linux.

STEP 1 The first step with ensuring Linux is ready for your C++ code is check the system and software are up to date. Open a Terminal and enter: `sudo apt-get update && sudo apt-get upgrade`. Press Return and enter your password. These commands updates the entire system and any installed software.


File
Edit
View
Search
Terminal
Help

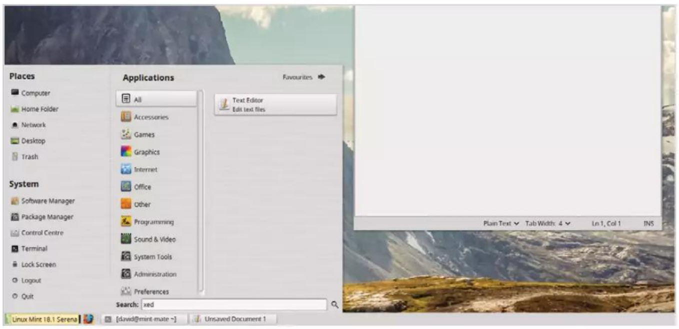
```
david@mint-mate ~ $ sudo apt-get update &
[sudo] password for david: 
```

STEP 2 Most Linux distros come preinstalled with all the necessary components to start coding in C++. However, it's always worth checking to see if everything is present, so still within the Terminal, enter: `sudo apt-get install build-essential` and press Return. If you have the right components, nothing is installed but if you're missing some then they are installed by the command.


david@mint-mate ~
File
Edit
View
Search
Terminal
Help

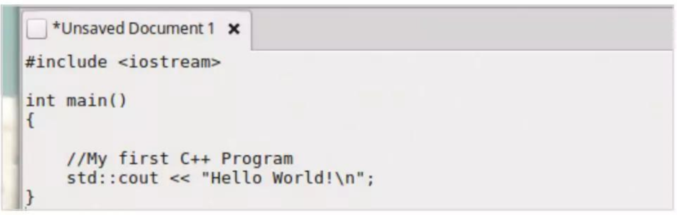
```
david@mint-mate ~ $ sudo apt-get install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.1ubuntu2).
build-essential set to manually installed.
0 to upgrade, 0 to newly install, 0 to remove and 0 not to upgrade.
david@mint-mate ~ $ 
```

STEP 3 Amazingly, that's it. Everything is all ready for you to start coding. Here's how to get your first C++ program up and running. In Linux Mint the main text editor is Xed can be launched by clicking on the Menu and typing `Xed` into the search bar. Click on the Text Editor button in the right-hand pane to open Xed.


Places
Computer
Home Folder
Network
Desktop
Trash
System
Software Manager
Package Manager
Control Centre
Terminal
Lock Screen
Logout
Quit
Applications
Favourites
All
Accessories
Games
Graphics
Internet
Office
Other
Programming
Sound & Video
System Tools
Administration
Preferences
Text Editor
Edit text files
Search: xed
Plain Text
Tab Width: 4
Ln 1, Col 1
INS
Linux Mint 18.1 Serena
[david@mint-mate ~]
Unsaved Document 1

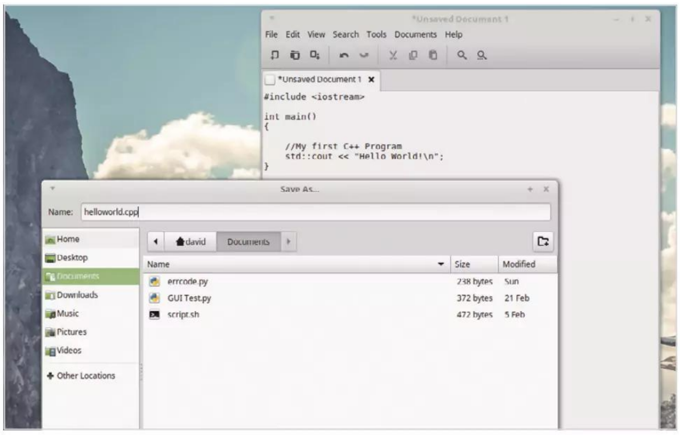
STEP 4 In Xed, or any other text editor you may be using, enter the lines of code that make up your C++ Hello World program. To remind you, it's:

```
#include <iostream>

int main()

{

//My first C++ program

std::cout <<  “Hello World!\n”;
```


*Unsaved Document 1

```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

STEP 5 When you've entered your code, click File > Save As and choose a folder where you want to save your program. Name the file as helloworld.cpp, or any other name just as long as it has .cpp as the extension. Click Save to continue.

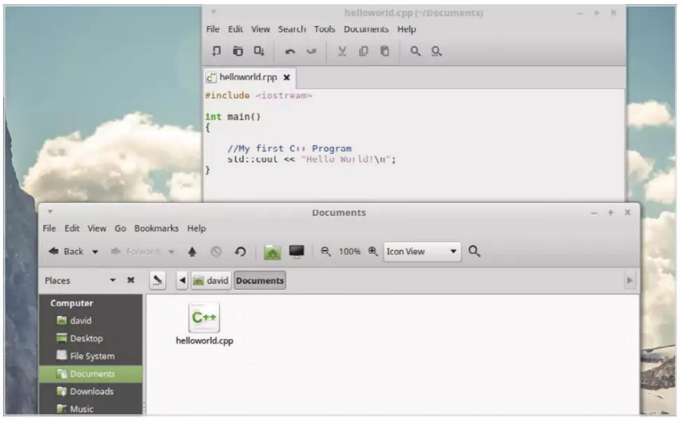

*Unsaved Document 1
File Edit View Search Tools Documents Help
*Unsaved Document 1
Save As...
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Other Locations
david
Documents

```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

Name: helloworld.cpp

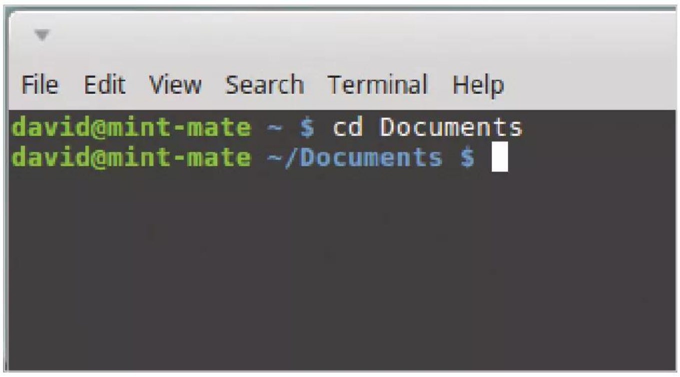
Name	Size	Modified
errcode.py	238 bytes	Sun
GUI Test.py	372 bytes	21 Feb
script.sh	472 bytes	5 Feb

STEP 6 The first thing you can see is that Xed has automatically recognised this as a C++ file, since the file extension is now set to .cpp. The colour coding is present in the code and if you open up the file manager you can also see that the file's icon has C++ stamped on it.


helloworld.cpp (~/Documents)
File Edit View Search Tools Documents Help
helloworld.cpp
Documents
File Edit View Go Bookmarks Help
Back
Forward
100%
Icon View
Places
david
Documents
Computer
david
Desktop
File System
Documents
Downloads
Music
C++
helloworld.cpp

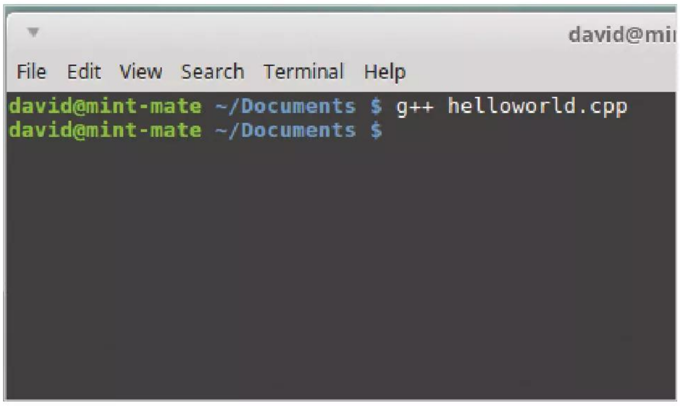
```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

STEP 7 With your code now saved, drop into the Terminal again. You need to navigate to the location of the C++ file you've just saved. Our example is in the Documents folder, so we can navigate to it by entering: `cd Documents`. Remember, the Linux Terminal is case sensitive, so any capitals must be entered correctly.


File Edit View Search Terminal Help

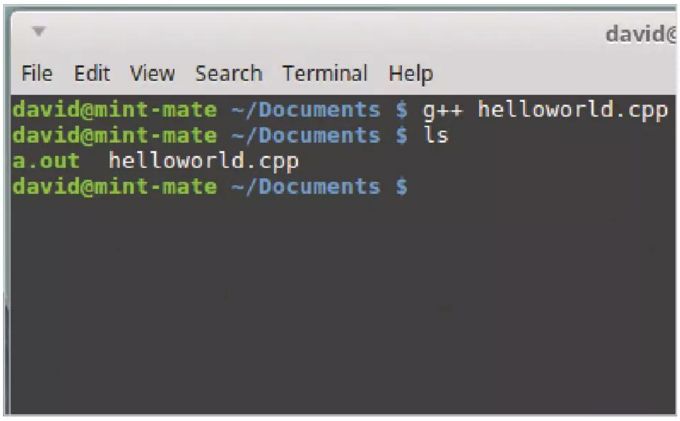
```
david@mint-mate ~ $ cd Documents
david@mint-mate ~/Documents $ 
```

STEP 8 Before you can execute the C++ file you need to compile it. In Linux it's common to use g++, an open source C++ compiler and as you're now in the same folder as the C++ file, go to the Terminal, enter: `g++ helloworld.cpp` and press return.


david@mi
File Edit View Search Terminal Help

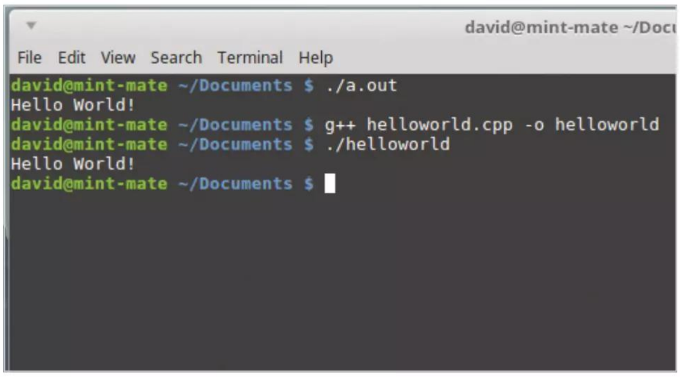
```
david@mint-mate ~/Documents $ g++ helloworld.cpp
david@mint-mate ~/Documents $
```

STEP 9 There will be a brief pause as the code is compiled by g++ and providing there are no mistakes or errors in the code you are returned to the command prompt. The compiling of the code has created a new file. If you enter `ls` into the Terminal you can see that alongside your C++ file is a.out.


david@
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ g++ helloworld.cpp
david@mint-mate ~/Documents $ ls
a.out  helloworld.cpp
david@mint-mate ~/Documents $
```

STEP 10 The a.out file is the compiled C++ code. To run the code enter: `./a.out` and press Return. The words 'Hello World!' appears on the screen. However, a.out isn't very friendly. To name it something else post-compiling, you can recompile with: `g++ helloworld.cpp -o helloworld`. This creates an output file called helloworld which can be run with: `./helloworld`.


david@mint-mate ~/Docu
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ ./a.out
Hello World!
david@mint-mate ~/Documents $ g++ helloworld.cpp -o helloworld
david@mint-mate ~/Documents $ ./helloworld
Hello World!
david@mint-mate ~/Documents $ 
```

Other C++ IDEs to Install

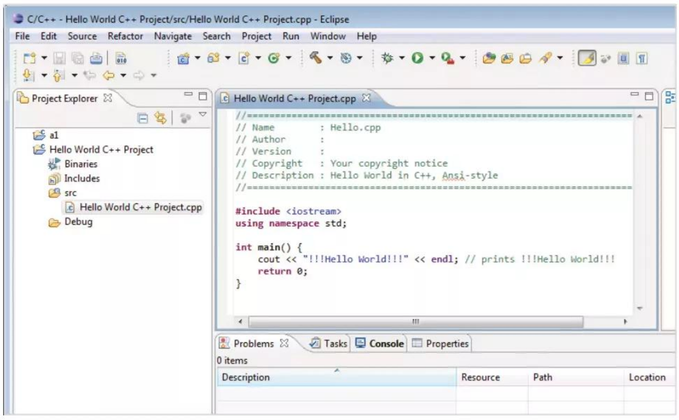
If you want to try a different approach to working with your C++ code, then there are plenty of options available to you. Windows is the most prolific platform for C++ IDEs but there are plenty for Mac and Linux users too.

DEVELOPING C++

Here are ten great C++ IDEs that are worth looking into. You can install one or all of them if you like, but find the one that works best for you.

ECLIPSE

Eclipse is a hugely popular C++ IDE that offers the programmer a wealth of features. It has a great, clean interface, is easy to use and available for Windows, Linux and Mac. Head over to **www.eclipse.org/downloads/** to download the latest version. If you're stuck, click the Need Help link for more information.

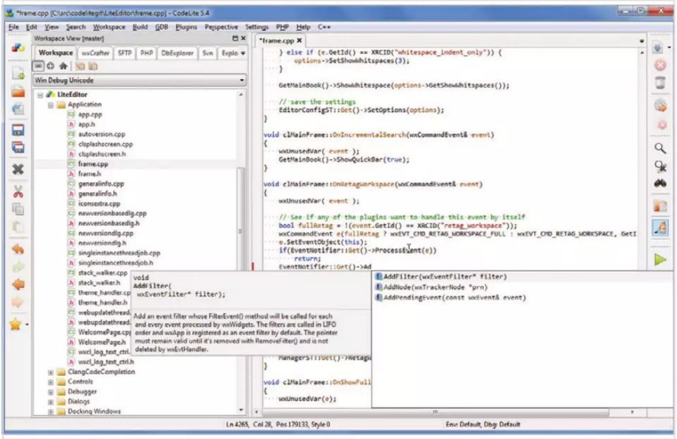


```
//==========================================================================
// Name        : Hello.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//==========================================================================

#include <iostream>
using namespace std;

int main() {
    cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
    return 0;
}
```

C/C++ - Hello World C++ Project/src/Hello World C++ Project.cpp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
a1
Hello World C++ Project
Binaries
Includes
src
Hello World C++ Project.cpp
Debug
Hello World C++ Project.cpp
Problems
Tasks
Console
Properties
0 items
Description
Resource
Path
Location

CODELITE

CodeLite is a free and open source IDE that's regularly updated and available for Windows, Linux and macOS. It's lightweight, uncomplicated and extremely powerful. You can find out more information as well as how to download and install it at **www.codelite.org/**.

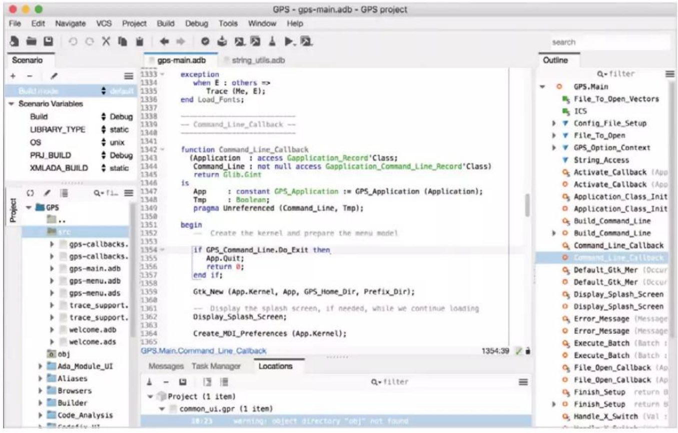


```
    } else if (e.GetId() == XRCID("whitespace_indent_only")) {
        options->SetShowWhitspaces(3);
    }

    GetMainBook()->ShowWhitespace(options->GetShowWhitspaces());

    // save the settings
    EditorConfigST::Get()->SetOptions(options);
}

void clMainFrame::OnIncrementalSearch(wxCommandEvent& event)
{
    wxUnusedVar( event );
    GetMainBook()->ShowQuickBar(true);
}

void clMainFrame::OnRetagWorkspace(wxCommandEvent& event)
{
    wxUnusedVar( event );

    // See if any of the plugins want to handle this event by itself
    bool fullRetag = !(event.GetId() == XRCID("retag_workspace"));
    wxCommandEvent e(fullRetag ? wxEVT_CMD_RETAG_WORKSPACE_FULL : wxEVT_CMD_RETAG_WORKSPACE, GetI
    e.SetEventObject(this);
    if(EventNotifier::Get()->ProcessEvent(e))
        return;
    EventNotifier::Get()->Ad
```

*frame.cpp
Workspace View [master]
Win Debug Unicode
LiteEditor
Application
AddFilter(wxEventFilter* filter)
AddNode(wxTrackerNode *prn)
AddPendingEvent(const wxEvent& event)
Ln 4265, Col 28, Pos 179133, Style 0
Env: Default, Dbg: Default

GNAT

The GNAT Programming Studio (GPS) is a powerful and intuitive IDE that supports testing, debugging and code analysis. The Community Edition is free, whereas the Pro version costs; however, the Community Edition is available for Windows, Mac, Linux and even the Raspberry Pi. You can find it at **www.adacore.com/download**.

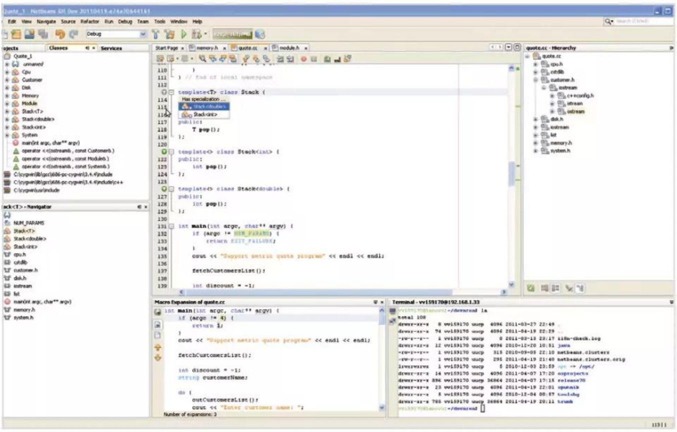


```
   exception
      when E : others =>
         Trace (Me, E);
   end Load_Fonts;

   --------------------------
   -- Command_Line_Callback --
   --------------------------

   function Command_Line_Callback
     (Application  : access Gapplication_Record'Class;
      Command_Line : not null access Gapplication_Command_Line_Record'Class)
      return Glib.Gint
   is
      App     : constant GPS_Application := GPS_Application (Application);
      Tmp     : Boolean;
      pragma Unreferenced (Command_Line, Tmp);

   begin
      --  Create the kernel and prepare the menu model

      if GPS_Command_Line.Do_Exit then
         App.Quit;
         return 0;
      end if;

      Gtk_New (App.Kernel, App, GPS_Home_Dir, Prefix_Dir);

      --  Display the splash screen, if needed, while we continue loading
      Display_Splash_Screen;

      Create_MDI_Preferences (App.Kernel);
```

GPS - gps-main.adb - GPS project
File Edit Navigate VCS Project Build Debug Tools Window Help
gps-main.adb
string_utils.adb
Scenario
Scenario Variables
Build: Debug
LIBRARY_TYPE: static
OS: unix
PRJ_BUILD: Debug
XMLADA_BUILD: static
GPS.Main.Command_Line_Callback
1354:39
Messages
Task Manager
Locations
Project (1 item)
common_ui.gpr (1 item)
Outline

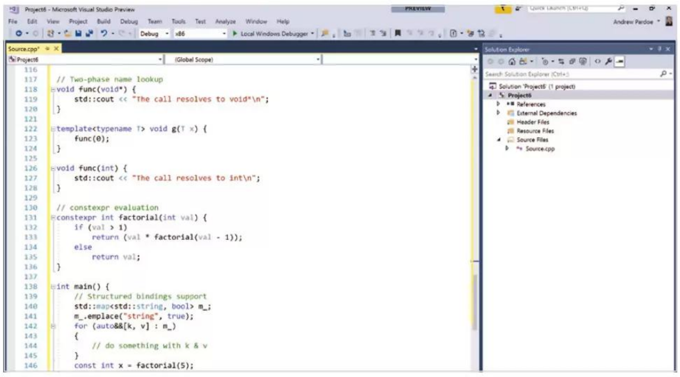
NETBEANS

Another popular choice is NetBeans. This is another excellent IDE that's packed with features and a pleasure to use. NetBeans IDE includes project based templates for C++ that give you the ability to build applications with dynamic and static libraries. Find out more at **www.netbeans.org/features/cpp/index.html.**



VISUAL STUDIO

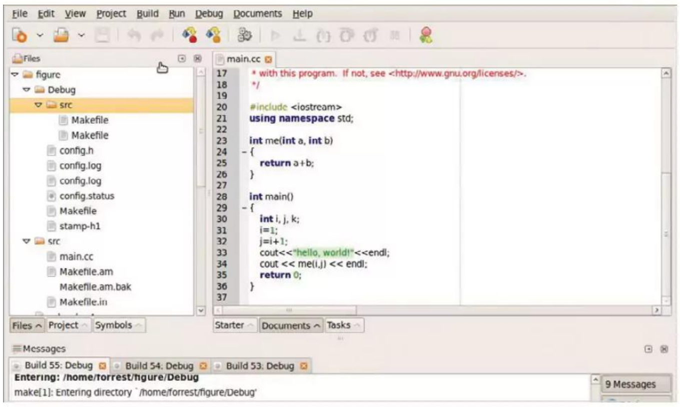
Microsoft's Visual Studio is a mammoth C++ IDE that allows you to create applications for Windows, Android, iOS and the web. The Community version is free to download and install but the other versions allow a free trial period. Go to **www.visualstudio.com/** to see what it can do for you.


Project6 - Microsoft Visual Studio Preview
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
x86
Local Windows Debugger
Source.cpp*
Project6
(Global Scope)
Solution Explorer

```
// Two-phase name lookup
void func(void*) {
    std::cout << "The call resolves to void*\n";
}

template<typename T> void g(T x) {
    func(0);
}

void func(int) {
    std::cout << "The call resolves to int\n";
}

// constexpr evaluation
constexpr int factorial(int val) {
    if (val > 1)
        return (val * factorial(val - 1));
    else
        return val;
}

int main() {
    // Structured bindings support
    std::map<std::string, bool> m_;
    m_.emplace("string", true);
    for (auto&&[k, v] : m_)
    {
        // do something with k & v
    }
    const int x = factorial(5);
```

ANJUTA

The Anjuta DevStudio is a Linux-only IDE that features some of the more advanced features you would normally find in a paid software development studio. There's a GUI designer, source editor, app wizard, interactive debugger and much more. Go to **www.anjuta.org/** for more information.

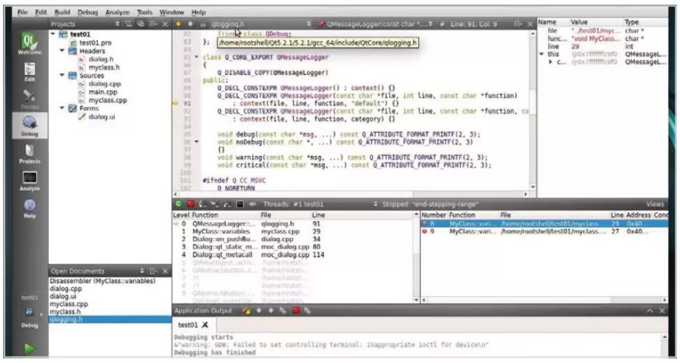

File Edit View Project Build Run Debug Documents Help
Files
figure
Debug
src
Makefile
Makefile
config.h
config.log
config.log
config.status
Makefile
stamp-h1
src
main.cc
Makefile.am
Makefile.am.bak
Makefile.in
Files
Project
Symbols
main.cc
Starter
Documents
Tasks
Messages
Build 55: Debug
Build 54: Debug
Build 53: Debug
9 Messages

```
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include <iostream>
using namespace std;

int me(int a, int b)
{
   return a+b;
}

int main()
{
   int i, j, k;
   i=1;
   j=i+1;
   cout<<"hello, world!"<<endl;
   cout << me(i,j) << endl;
   return 0;
}
```

```
Entering: /home/forrest/figure/Debug
make[1]: Entering directory `/home/forrest/figure/Debug'
```

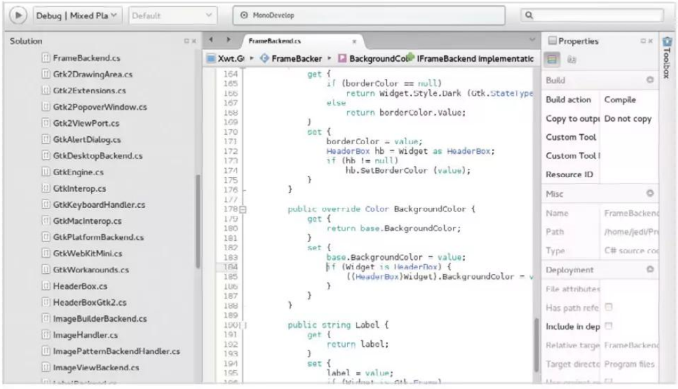
QT CREATOR

This cross-platform IDE is designed to create C++ applications for desktop and mobile environments. It comes with a code editor and integrated tools for testing and debugging, as well as deploying to you chosen platform. It's not free but there is a trial period on offer before requiring purchasing: **www.qt.io/qt-features-libraries-apis-tools-and-ide/**.


File Edit Build Debug Analyze Tools Window Help
Projects
test01
test01.pro
Headers
dialog.h
myclass.h
Sources
dialog.cpp
main.cpp
myclass.cpp
Forms
dialog.ui
Welcome
Edit
Design
Debug
Projects
Analyze
Help
qlogging.h
Open Documents
Application Output
test01
Debug

MONODEVELOP

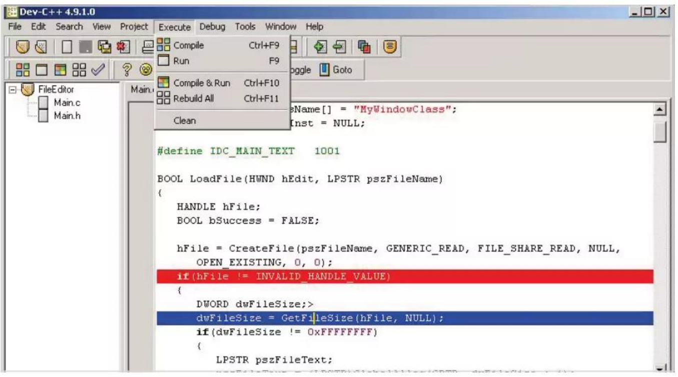
This excellent IDE allows developers to write C++ code for desktop and web applications across all the major platforms. There's an advanced text editor, integrated debugger and a configurable workbench to help you create your code. It's available for Windows, Mac and Linux and is free to download and use: **www.monodevelop.com/**.


Debug | Mixed Pla
Default
MonoDevelop
Solution
FrameBackend.cs
Gtk2DrawingArea.cs
Gtk2Extensions.cs
Gtk2PopoverWindow.cs
Gtk2ViewPort.cs
GtkAlertDialog.cs
GtkDesktopBackend.cs
GtkEngine.cs
GtkInterop.cs
GtkKeyboardHandler.cs
GtkMacInterop.cs
GtkPlatformBackend.cs
GtkWebKitMini.cs
GtkWorkarounds.cs
HeaderBox.cs
HeaderBoxGtk2.cs
ImageBuilderBackend.cs
ImageHandler.cs
ImagePatternBackendHandler.cs
ImageViewBackend.cs
Xwt.G
FrameBacker
BackgroundCol
IFrameBackend implementatic
Properties
Build
Build action
Compile
Copy to outpu
Do not copy
Custom Tool
Custom Tool
Resource ID
Misc
Name
Path
Type
Deployment
File attributes
Has path refe
Include in dep
Relative targe
Target directo
Program files
Toolbox

```
        get {
            if (borderColor == null)
                return Widget.Style.Dark (Gtk.StateType
            else
                return borderColor.Value;
        }
        set {
            borderColor = value;
            HeaderBox hb = Widget as HeaderBox;
            if (hb != null)
                hb.SetBorderColor (value);
        }
    }

    public override Color BackgroundColor {
        get {
            return base.BackgroundColor;
        }
        set {
            base.BackgroundColor = value;
            if (Widget is HeaderBox) {
                ((HeaderBox)Widget).BackgroundColor = v
            }
        }
    }

    public string Label {
        get {
            return label;
        }
        set {
            label = value;
```

DEV C++

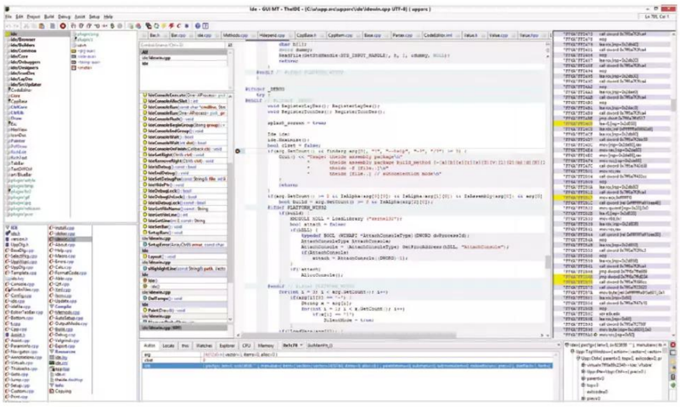
Bloodshed Dev C++, despite its colourful name, is an older IDE that is for Windows systems only. However, many users praise its clean interface and uncomplicated way of coding and compiling. Although there's not been much updating for some time, it's certainly one to consider if you want something different: **www.bloodshed.net/devcpp.html**.


Dev-C++ 4.9.1.0
File Edit Search View Project Execute Debug Tools Window Help
Compile Ctrl+F9
Run F9
Compile & Run Ctrl+F10
Rebuild All Ctrl+F11
Clean
oggle
Goto
FileEditor
Main.c
Main.h
Main.

```
sName[] = "MyWindowClass";
Inst = NULL;

#define IDC_MAIN_TEXT   1001

BOOL LoadFile(HWND hEdit, LPSTR pszFileName)
{
   HANDLE hFile;
   BOOL bSuccess = FALSE;

   hFile = CreateFile(pszFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
      OPEN_EXISTING, 0, 0);
   if(hFile != INVALID_HANDLE_VALUE)
   {
      DWORD dwFileSize;>
      dwFileSize = GetFileSize(hFile, NULL);
      if(dwFileSize != 0xFFFFFFFF)
      {
         LPSTR pszFileText;
```

U++

Ultimate++ is a cross-platform C++ IDE that boats a rapid development of code through the smart and aggressive use of C++. For the novice, it's a beast of an IDE but behind its complexity is a beauty that would make a developer's knees go wobbly. Find out more at **www.ultimatepp.org/index.html**.



This section contains the building block of C++, from learning how to compile and execute your first C++ code, through to developing user interaction. There's a lot to learn with C++, but these tutorials, tips and tricks will lead you in the right direction.

C++ is different to the other programming languages in this book, and it's continually evolving as the applications that require it are forever improving. If you know C++, then you're well on your way to becoming a much sought-after coder and a vital member of the coding community.

36 Your First C++ Program
38 Compile and Execute
40 Using Comments
42 Variables
44 Data Types
46 Strings
48 C++ Maths
50 User Interaction

Coding with C++

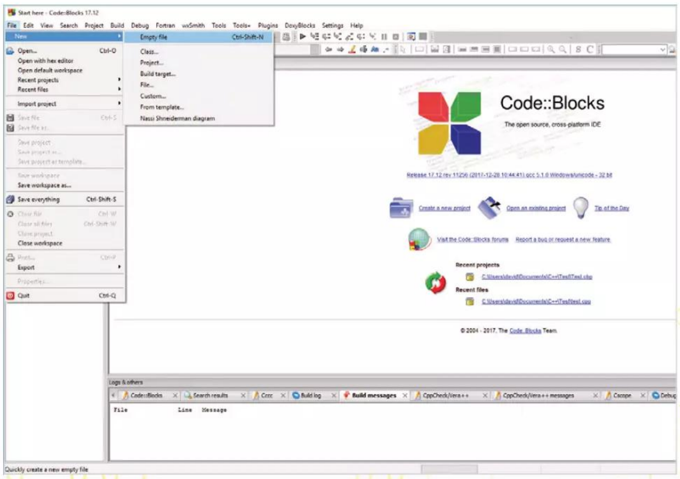
Your First C++ Program

You may have followed the Mac and Linux examples previously but you're going to be working exclusively in Windows and Code::Blocks from here on. Let's begin by writing your first C++ program and taking the first small step into a larger coding world.

HELLO, WORLD!

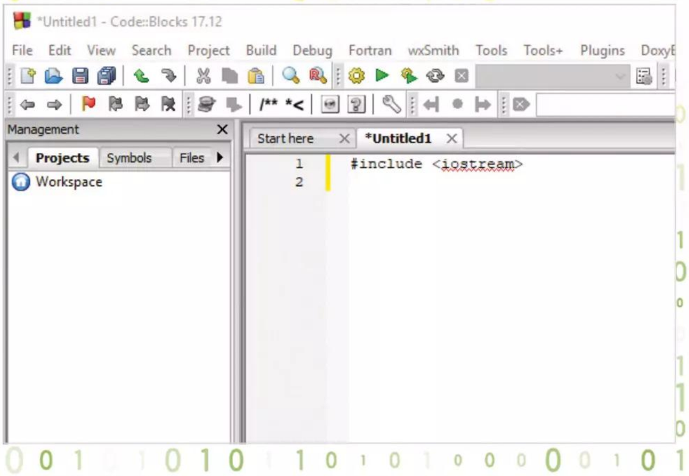
It's traditional in programming for the first code to be entered to output the words 'Hello, World!' to the screen. Interestingly, this dates back to 1968 using a language called BCPL.

STEP 1 We're going to use Windows 10 and the excellent IDE Code::Blocks for the rest of the C++ code and tutorials in this book. Begin by launching Code::Blocks. When open, click on File > New > Empty File or press Ctrl+Shift+N on the keyboard.


Start here - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
New
Empty file Ctrl-Shift-N
Class...
Project...
Build target...
File...
Custom...
From template...
Nassi Shneiderman diagram
Open... Ctrl-O
Open with hex editor
Open default workspace
Recent projects
Recent files
Import project
Save file
Save everything
Close workspace
Print...
Export
Quit Ctrl-Q
Code::Blocks
The open source, cross-platform IDE
Create a new project
Open an existing project
Tip of the Day
Recent projects
Recent files
Logs & others
Build messages
Quickly create a new empty file

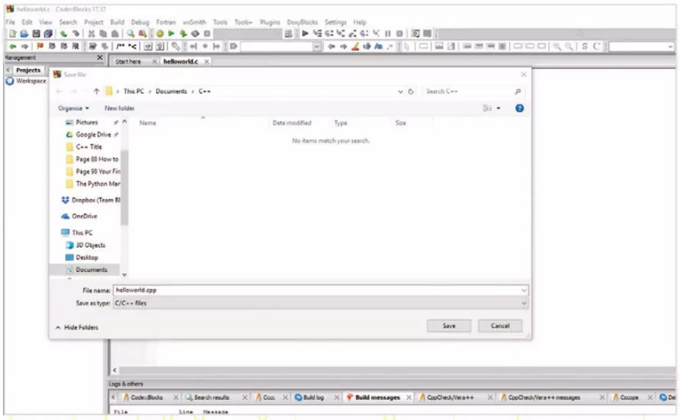
STEP 2 Now you can see a blank screen, with the tab labelled *Untitled1, and the number one in the top left of the main Code::Blocks window. Begin by clicking in the main window, so the cursor is next to the number one, and entering:

```
#include <iostream>
```


*Untitled1 - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins Doxy
Management
Projects Symbols Files
Workspace
Start here
*Untitled1

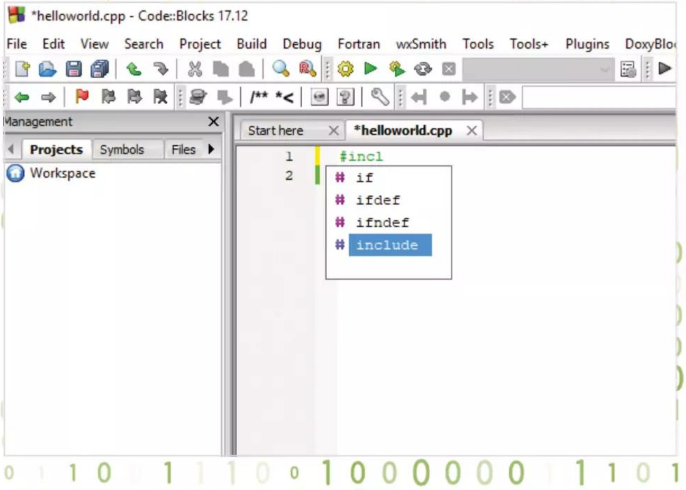
```
#include <iostream>
```

STEP 3 At the moment it doesn't look like much, and it makes even less sense, but we'll get to that in due course. Now click on File > Save File As. Create or find a suitable location on your hard drive and in the File Name box, call it helloworld.cpp. Click the Save as type box and select C/C++ files. Click the Save button.


Save file
This PC > Documents > C++
Organise New folder
Name Date modified Type Size
No items match your search.
Save
Cancel
Hide Folders

File name: helloworld.cpp
Save as type: C/C++ files

STEP 4 You can see that Code::Blocks has now changed the colour coding, recognising that the file is now C++ code. This means that code can be auto-selected from the Code::Blocks repository. Delete the #include <iostream> line and re-enter it. You can see the auto-select boxes appearing.


*helloworld.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlo
Management
Projects Symbols Files
Workspace
Start here
*helloworld.cpp
# if
# ifdef
# ifndef
# include

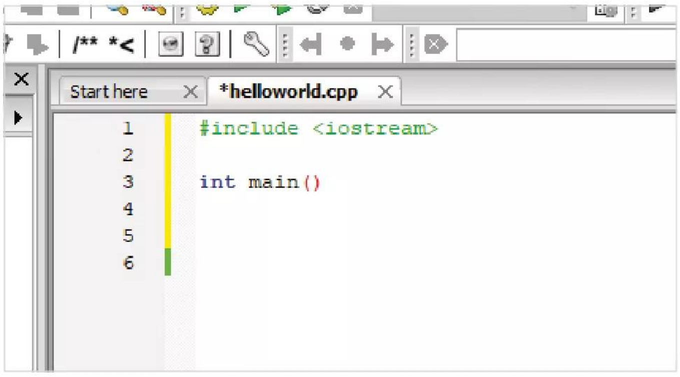
```
#incl
```

STEP 5

Auto-selection of commands is extremely handy and cuts out potential mistyping. Press Return to get to line 3, then enter:

`int main()`

Note: there's no space between the brackets.


Start here
*helloworld.cpp

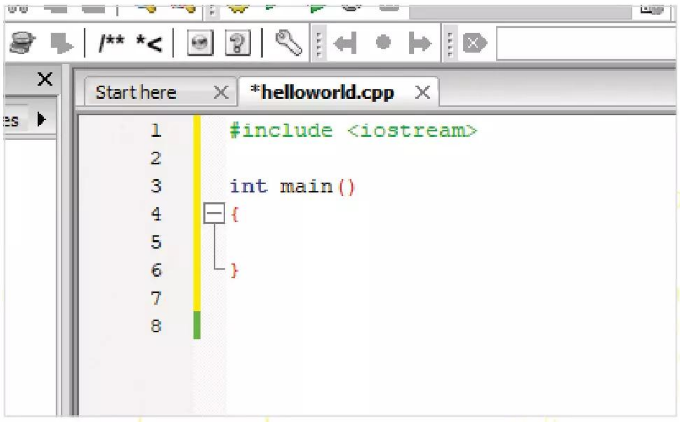
```
1  #include <iostream>
2
3  int main()
4
5
6
```

STEP 6

On the next line below int main(), enter a curly bracket:

`{`

This can be done by pressing Shift and the key to the right of P on an English UK keyboard layout.

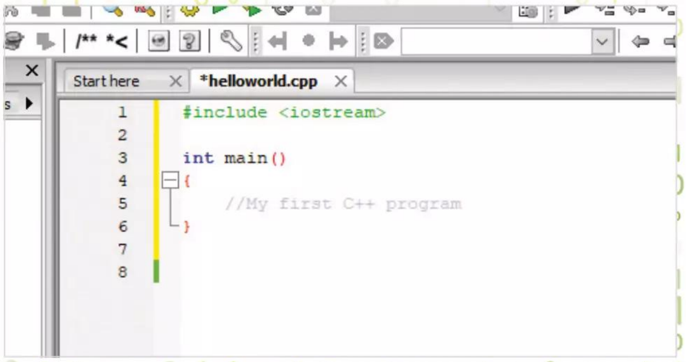

Start here
*helloworld.cpp

```
1  #include <iostream>
2
3  int main()
4  {
5
6  }
7
8
```

STEP 7

Notice that Code::Blocks has automatically created a corresponding closing curly bracket a couple of lines below, linking the pair, as well as a slight indent. This is due to the structure of C++ and it's where the meat of the code is entered. Now enter:

`//My first C++ program`

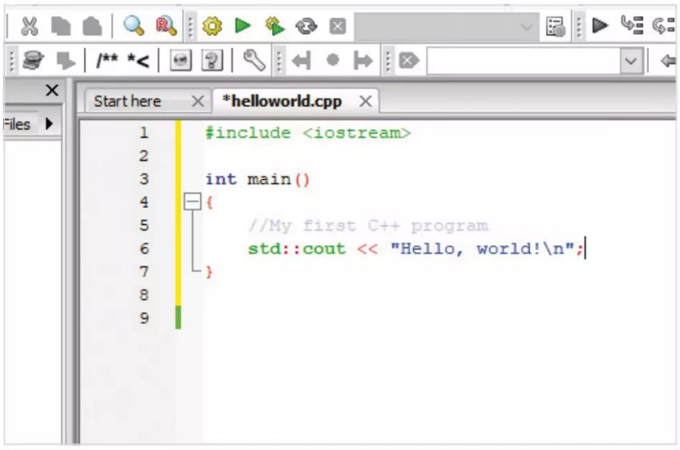

Start here
*helloworld.cpp

```
1  #include <iostream>
2
3  int main()
4  {
5      //My first C++ program
6  }
7
8
```

STEP 8

Note again the colour coding change. Press Return at the end of the previous step's line, and then enter:

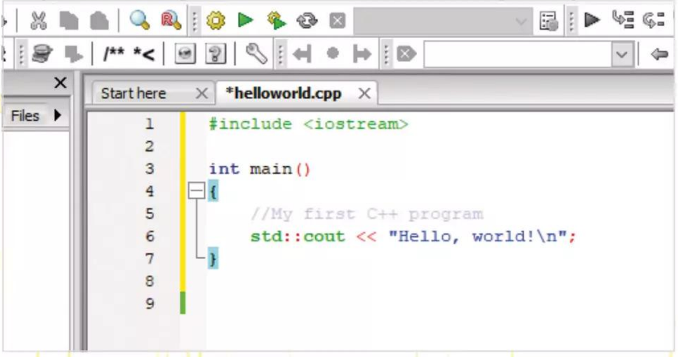
`std::cout << “Hello, world!\n”;`


Start here
*helloworld.cpp
Files

```
1  #include <iostream>
2
3  int main()
4  {
5      //My first C++ program
6      std::cout << "Hello, world!\n";
7  }
8
9
```

STEP 9

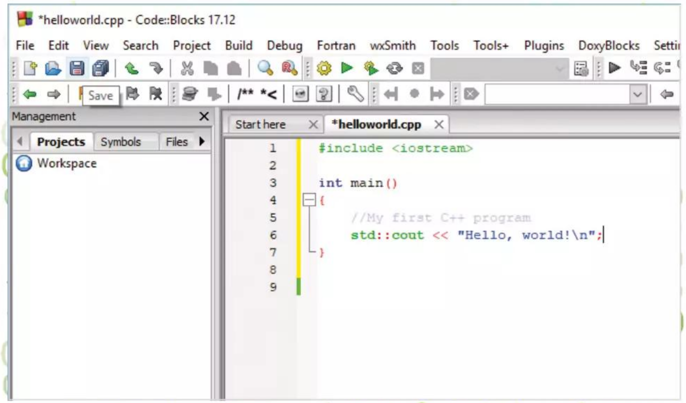
Just as before, Code::Blocks auto-completes the code you're entering, including placing a closing speech mark as soon as you enter the first. Don't forget the semicolon at the end of the line; this is one of the most important elements to a C++ program and we'll tell you why in the next section. For now, move the cursor down to the closing curly bracket and press Return.


Start here
*helloworld.cpp
Files

```
1  #include <iostream>
2
3  int main()
4  {
5      //My first C++ program
6      std::cout << "Hello, world!\n";
7  }
8
9
```

STEP 10

That's all you need to do for the moment. It may not look terribly amazing but C++ is best absorbed in small chunks. Don't execute the code at the moment as you need to look at how a C++ program is structured first; then you can build and run the code. For now, click on Save, the single floppy disc icon.


*helloworld.cpp - Code::Blocks 17.12
File
Edit
View
Search
Project
Build
Debug
Fortran
wxSmith
Tools
Tools+
Plugins
DoxyBlocks
Setti
Save
Management
Projects
Symbols
Files
Workspace
Start here
*helloworld.cpp

```
1  #include <iostream>
2
3  int main()
4  {
5      //My first C++ program
6      std::cout << "Hello, world!\n";
7  }
8
9
```

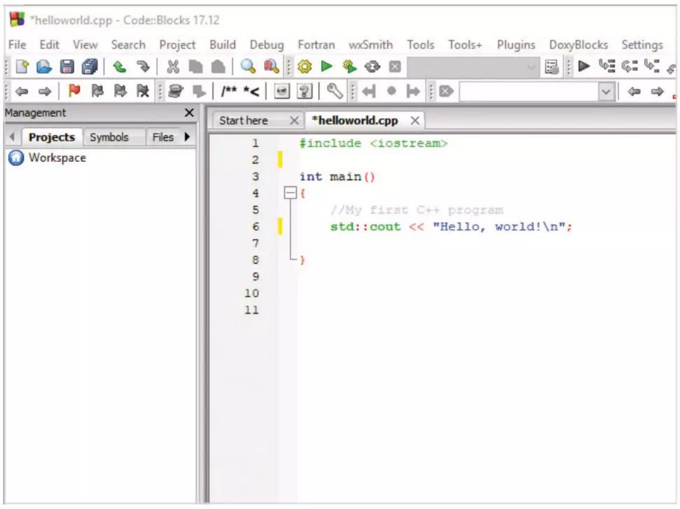
Compile and Execute

You've created your first C++ program and you now understand the basics behind the structure of one. Let's actually get things moving and compile and execute, or run if you prefer, the program and see how it looks.

GREETINGS FROM C++

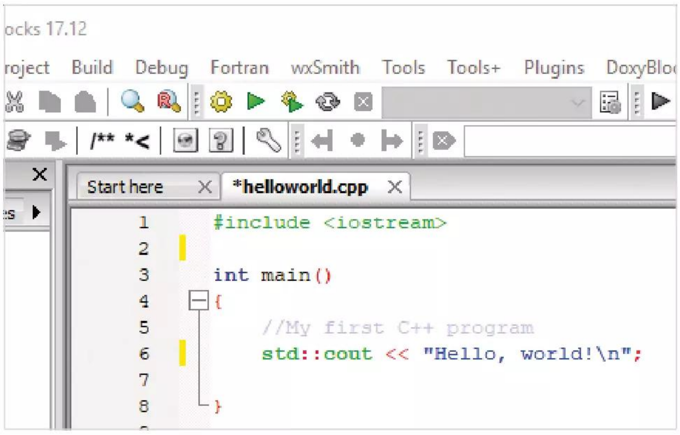
Compiling and executing C++ code from Code::Blocks is extraordinarily easy; it's just a matter of clicking an icon and seeing the result. Here's how it's done.

STEP 1 Open Code::Blocks, if you haven't already, and load up the previously saved Hello World code you created. Ensure that there are no visible errors, such as missing semicolons at the end of the std::cout line.


*helloworld.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings
Management
Projects Symbols Files
Workspace
Start here
*helloworld.cpp

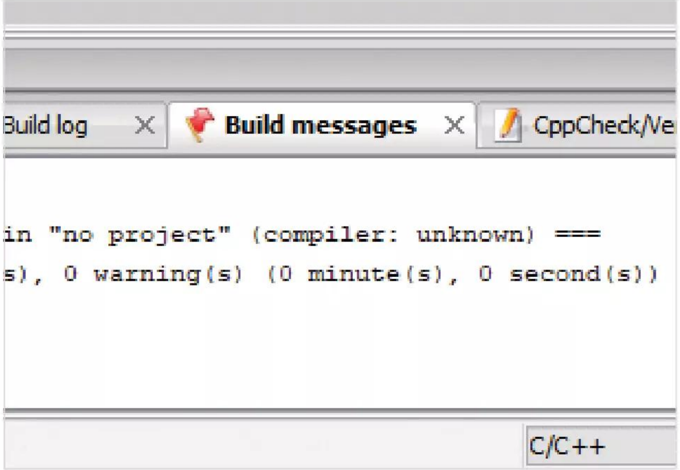
```
#include <iostream>

int main()
{
    //My first C++ program
    std::cout << "Hello, world!\n";
}
```

STEP 2 If your code is looking similar to the one in our screenshot, then look to the menu bar along the top of the screen. Under the Fortran entry in the topmost menu you can see a group of icons: a yellow cog, green play button and a cog/play button together. These are Build, Run, Build and Run functions.


ocks 17.12
roject Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBloc
Start here
*helloworld.cpp

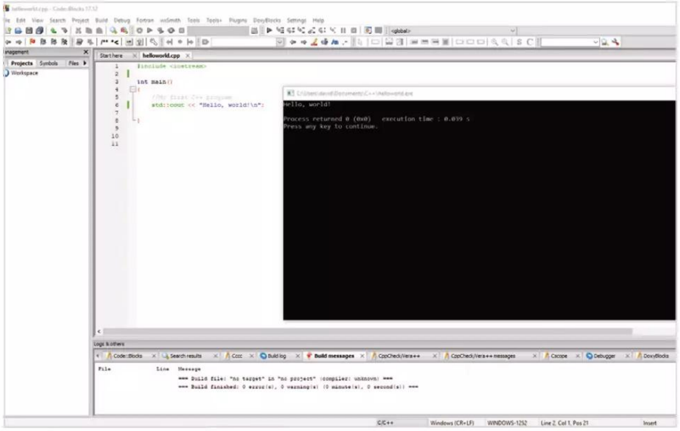
```
#include <iostream>

int main()
{
    //My first C++ program
    std::cout << "Hello, world!\n";
}
```

STEP 3 Start by clicking on the Build icon, the yellow cog. At this point, your code has now been run through the Code::Blocks compiler and checked for any errors. You can see the results of the Build by looking to the bottom window pane. Any messages regarding the quality of the code are displayed here.


Build log
Build messages
CppCheck/Ve
C/C++

```
in "no project" (compiler: unknown) ===
s), 0 warning(s) (0 minute(s), 0 second(s))
```

STEP 4 Now click on the Run icon, the green play button. A command line box appears on your screen displaying the words: Hello, world!, followed by the time it's taken to execute the code, and asking you press a key to continue. Well done, you just compiled and executed your first C++ program.


Logs & others
Code::Blocks
Search results
Cccc
Build log
Build messages
CppCheck/Vera++
C/C++
Windows (CR+LF)

```
Hello, world!

Process returned 0 (0x0)   execution time : 0.039 s
Press any key to continue.
```

```
File          Line   Message
                     === Build file: "no target" in "no project" (compiler: unknown) ===
                     === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

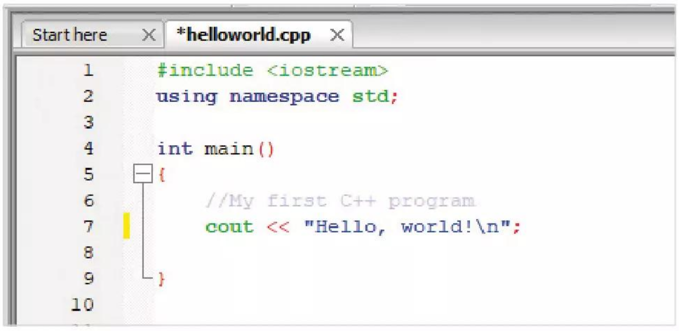
STEP 5

Pressing any key in the command line box closes it, returning you to Code::Blocks. Let's alter the code slightly. Under the #include line, enter:

```
using namespace std;
```

Then, delete the std:: part of the Cout line; like so:

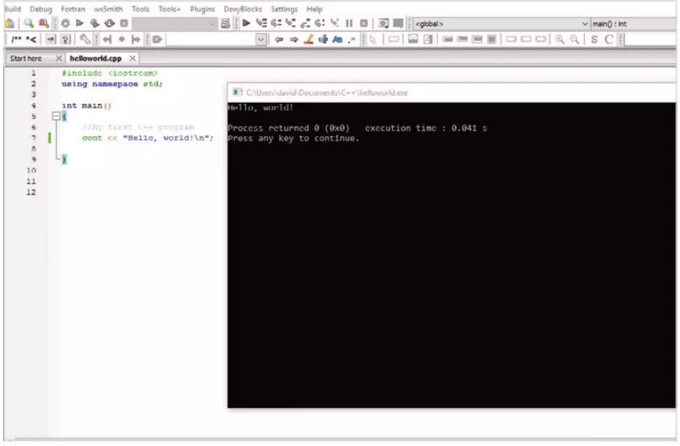
```
cout << "Hello, world\n";
```



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
}
```

STEP 6

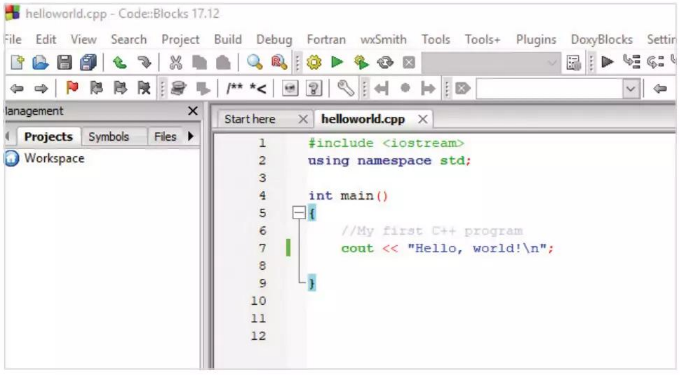
In order to apply the new changes to the code, you need to re-compile, build, and run it again. This time, however, you can simply click the Build/Run icon, the combined yellow cog and green play button.



```
Hello, world!

Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

STEP 7

Just as we mentioned in the previous pages, you don't need to have std::cout if you already declare using namespace std; at the beginning of the code. We could have easily clicked the Build/Run icon to begin with but it's worth going through the available options. You can also see that by building and running, the file has been saved.

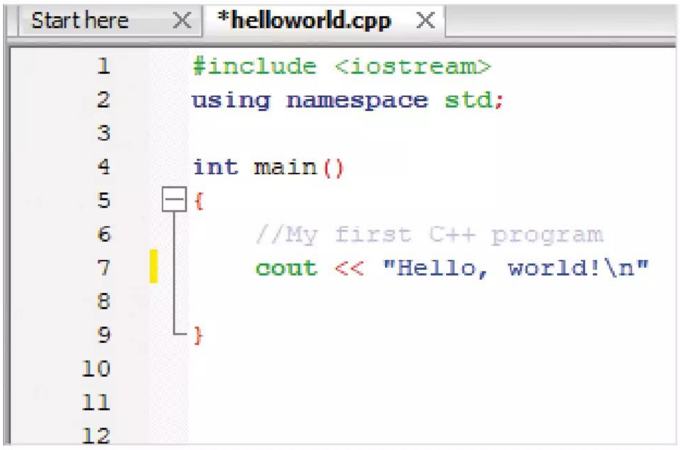

helloworld.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settir
Management
Projects Symbols Files
Workspace
Start here
helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
}
```

STEP 8

Create a deliberate error in the code. Remove the semicolon from the cout line, so it reads:

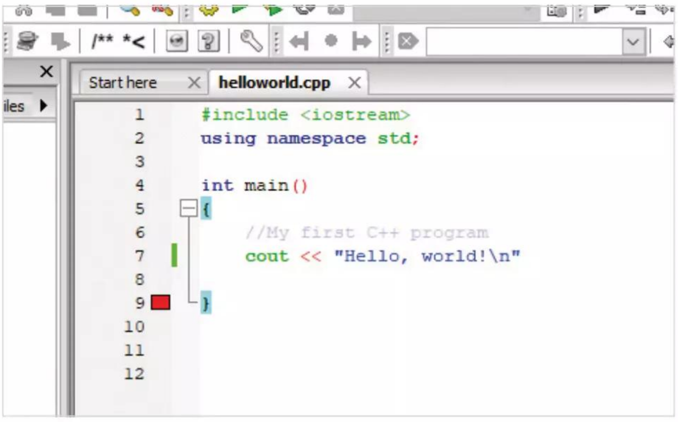
```
cout << "Hello, world!\n"
```



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n"
}
```

STEP 9

Now click the Build and Run icon again to apply the changes to the code. This time Code::Blocks refuses to execute the code, due to the error you put in. In the Log pane at the bottom of the screen you are informed of the error, in this case: Expected ';' before '}' token, indicating the missing semicolon.



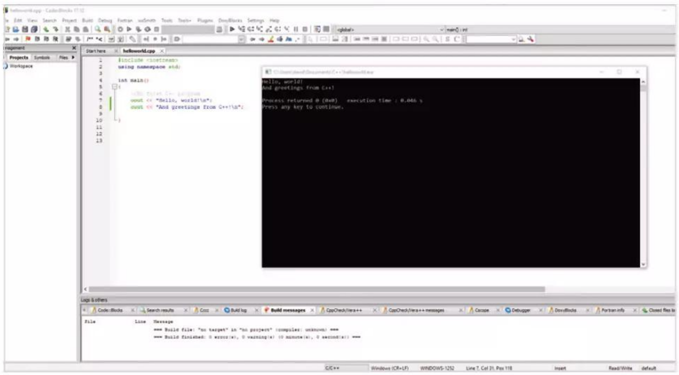
```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n"
}
```

STEP 10

Replace the semicolon and under the cout line, enter a new line to your code:

```
cout << "And greetings from C++!\n";
```

The \n simply adds a new line under the last line of outputted text. Build and Run the code, to display your handiwork.



```
Hello, world!
And greetings from C++!

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

Using Comments

While comments may seem like a minor element to the many lines of code that combine to make a game, application or even an entire operating system, in actual fact they're probably one of the most important factors.

THE IMPORTANCE OF COMMENTING

Comments inside code are basically human readable descriptions that detail what the code is doing at that particular point. They don't sound especially important but code without comments is one of the many frustrating areas of programming, regardless of whether you're a professional or just starting out.

In short, all code should be commented in such a manner as to effectively describe the purpose of a line, section, or individual elements. You should get in to the habit of commenting as much as possible, by imagining that someone who doesn't know anything about programming can pick up your code and understand what it's going to do simply by reading your comments.

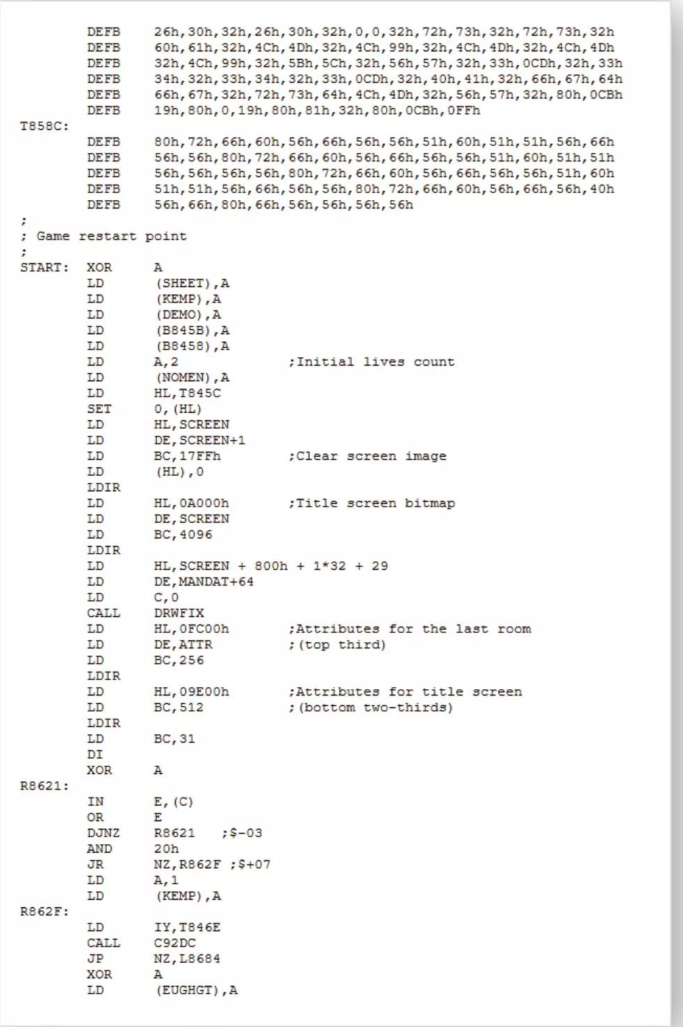
In a professional environment, comments are vital to the success of the code and ultimately, the company. In an organisation, many programmers work in teams alongside engineers, other developers, hardware analysts and so on. If you're a part of the team that's writing a bespoke piece of software for the company, then your comments help save a lot of time should something go wrong, and another team member has to pick up and follow the trail to pinpoint the issue.

Place yourself in the shoes of someone whose job it is to find out what's wrong with a program. The program has in excess of 800,000 lines of code, spread across several different modules. You can soon appreciate the need for a little help from the original programmers in the form of a good comment.

The best comments are always concise and link the code logically, detailing what happens when the program hits this line or section. You don't need to comment on every line. Something along the lines of: if x==0 doesn't require you to comment that if x equals zero then do something; that's going to be obvious to the reader. However, if x equalling zero is something that drastically changes the program for the user, such as, they've run out of lives, then it certainly needs to commented on.

Even if the code is your own, you should write comments as if you were going to publicly share it with others. This way you can return to that code and always understand what it was you did or where it was you went wrong or what worked brilliantly.

Comments are good practise and once you understand how to add a comment where needed, you soon do it as if it's second nature.



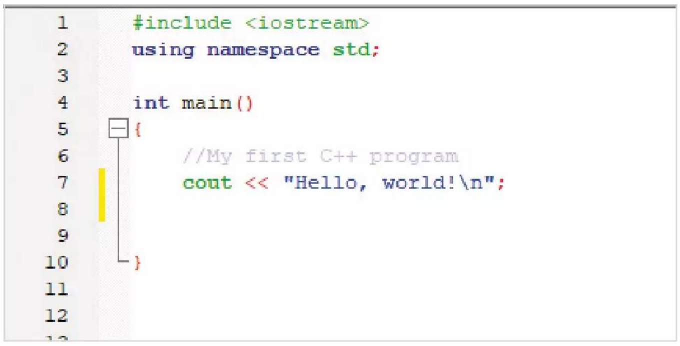
```
        DEFB    26h,30h,32h,26h,30h,32h,0,0,32h,72h,73h,32h,72h,73h,32h
        DEFB    60h,61h,32h,4Ch,4Dh,32h,4Ch,99h,32h,4Ch,4Dh,32h,4Ch,4Dh
        DEFB    32h,4Ch,99h,32h,5Bh,5Ch,32h,56h,57h,32h,33h,0CDh,32h,33h
        DEFB    34h,32h,33h,34h,32h,33h,0CDh,32h,40h,41h,32h,66h,67h,64h
        DEFB    66h,67h,32h,72h,73h,64h,4Ch,4Dh,32h,56h,57h,32h,80h,0CBh
        DEFB    19h,80h,0,19h,80h,81h,32h,80h,0CBh,0FFh
T858C:
        DEFB    80h,72h,66h,60h,56h,66h,56h,56h,51h,60h,51h,51h,56h,66h
        DEFB    56h,56h,80h,72h,66h,60h,56h,66h,56h,56h,51h,60h,51h,51h
        DEFB    56h,56h,56h,56h,80h,72h,66h,60h,56h,66h,56h,56h,51h,60h
        DEFB    51h,51h,56h,66h,56h,56h,80h,72h,66h,60h,56h,66h,56h,40h
        DEFB    56h,66h,80h,66h,56h,56h,56h,56h
;
; Game restart point
;
START:  XOR     A
        LD      (SHEET),A
        LD      (KEMP),A
        LD      (DEMO),A
        LD      (B845B),A
        LD      (B8458),A
        LD      A,2             ;Initial lives count
        LD      (NOMEN),A
        LD      HL,T845C
        SET     0,(HL)
        LD      HL,SCREEN
        LD      DE,SCREEN+1
        LD      BC,17FFh        ;Clear screen image
        LD      (HL),0
        LDIR
        LD      HL,0A000h       ;Title screen bitmap
        LD      DE,SCREEN
        LD      BC,4096
        LDIR
        LD      HL,SCREEN + 800h + 1*32 + 29
        LD      DE,MANDAT+64
        LD      C,0
        CALL    DRWFIX
        LD      HL,0FC00h       ;Attributes for the last room
        LD      DE,ATTR         ;(top third)
        LD      BC,256
        LDIR
        LD      HL,09E00h       ;Attributes for title screen
        LD      BC,512          ;(bottom two-thirds)
        LDIR
        LD      BC,31
        DI
        XOR     A
R8621:
        IN      E,(C)
        OR      E
        DJNZ    R8621   ;$-03
        AND     20h
        JR      NZ,R862F ;$+07
        LD      A,1
        LD      (KEMP),A
R862F:
        LD      IY,T846E
        CALL    C92DC
        JP      NZ,L8684
        XOR     A
        LD      (EUGHGT),A
```

C++ COMMENTS

Commenting in C++ involves using a double forward slash '/', or a forward slash and an asterisk, '/*'. You've already seen some brief examples but this is how they work.

STEP 1 Using the Hello World code as an example, you can easily comment on different sections of the code using the double forward slash:

```
//My first C++ program
cout << "Hello, world!\n";
```

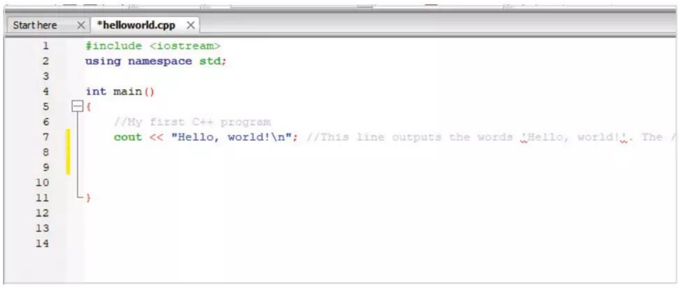


```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";


}
```

STEP 2 However, you can also add comments to the end of a line of code, to describe in a better way what's going on:

```
cout << "Hello, world!\n"; //This line outputs the
words 'Hello, world!'. The \n denotes a new line.
```

Note, you don't have to put a semicolon at the end of a comment. This is because it's a line in the code that's ignored by the compiler.

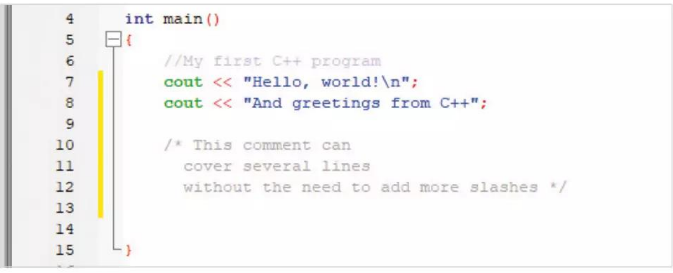


```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n"; //This line outputs the words 'Hello, world!'. The /



}
```

STEP 3 You can comment out several lines by using the forward slash and asterisk:

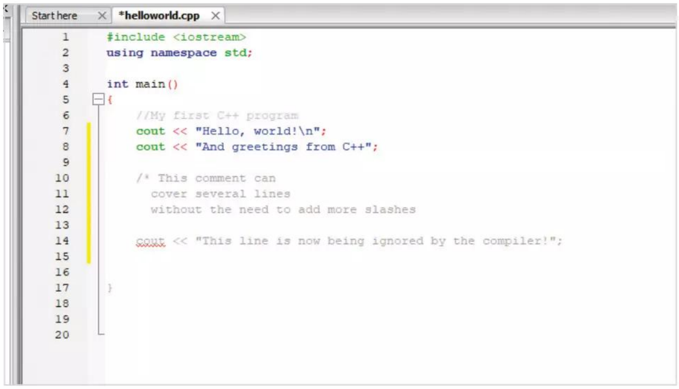
```
/* This comment can
   cover several lines
   without the need to add more slashes */
```

Just remember to finish the block comment with the opposite asterisk and forward slash.



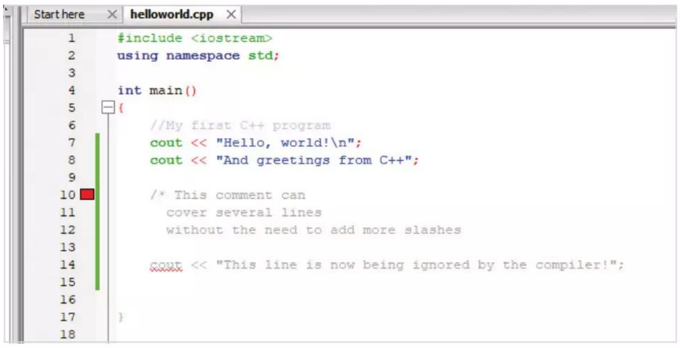
```
int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++";

    /* This comment can
      cover several lines
      without the need to add more slashes */


}
```

STEP 4 Be careful when commenting, especially with block comments. It's very easy to forget to add the closing asterisk and forward slash and thus negate any code that falls inside the comment block.



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++";

    /* This comment can
      cover several lines
      without the need to add more slashes

    cout << "This line is now being ignored by the compiler!";


}
```

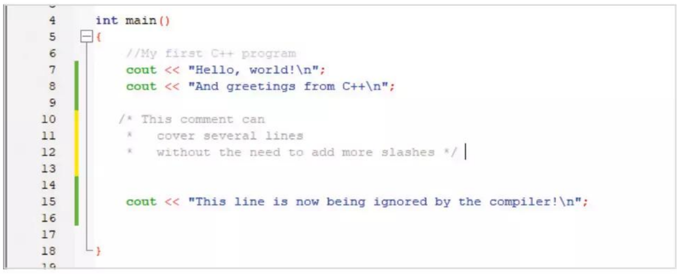
STEP 5 Obviously if you try and build and execute the code it errors out, complaining of a missing curly bracket '}' to finish off the block of code. If you've made the error a few times, then it can be time consuming to go back and rectify. Thankfully, the colour coding in Code::Blocks helps identify comments from code.



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++";

    /* This comment can
      cover several lines
      without the need to add more slashes

    cout << "This line is now being ignored by the compiler!";


}
```

STEP 6 If you're using block comments, it's good practise in C++ to add an asterisk to each new line of the comment block. This also helps you to remember to close the comment block off before continuing with the code:

```
/* This comment can
 *    cover several lines
 *    without the need to add more slashes */
```



```
int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++\n";

   /* This comment can
    *   cover several lines
    *   without the need to add more slashes */


    cout << "This line is now being ignored by the compiler!\n";


}
```

Variables

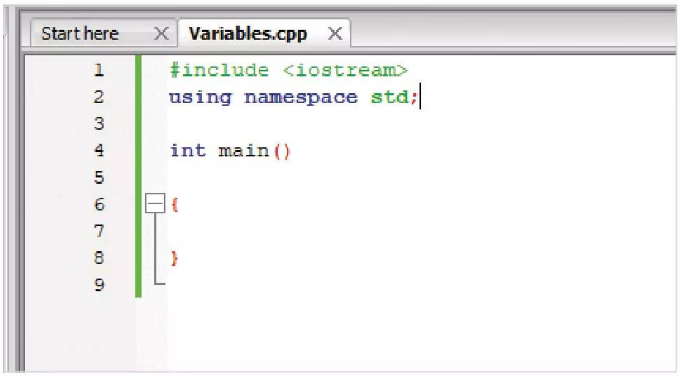
Variables differ slightly when using C++ as opposed to Python. In Python, you can simply state that ‘a’ equals 10 and a variable is assigned. However, in C++ a variable has to be declared with its type before it can be used.

THE DECLARATION OF VARIABLES

You can declare a C++ variable by using statements within the code. There are several distinct types of variables you can declare. Here’s how it works.

STEP 1 Open up a new, blank C++ file and enter the usual code headers:

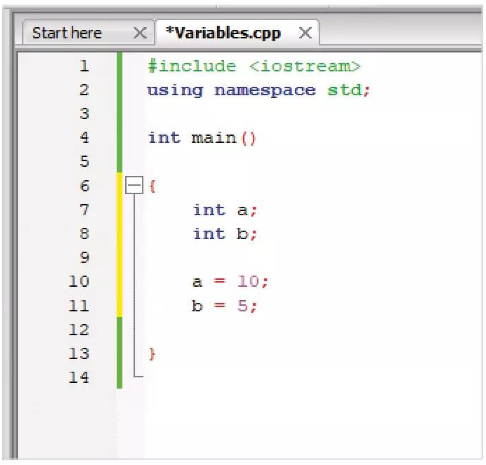
```
#include <iostream>
using namespace std;

int main()
{

}
```



```
#include <iostream>
using namespace std;

int main()

{

}

```

STEP 2 Start simple by creating two variables, a and b, with one having a value of 10 and the other 5. You can use the data type int to declare these variables. Within the curly brackets, enter:

```
int a;
int b;

a = 10;
b = 5;
```

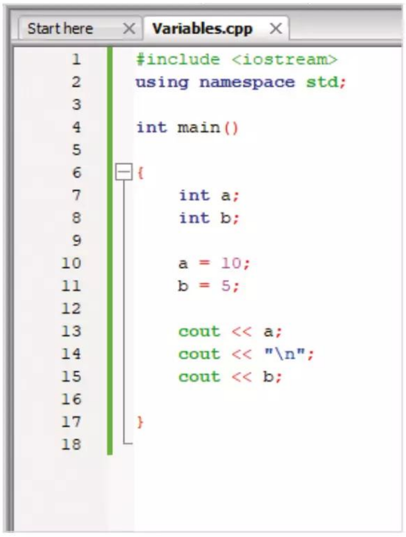


```
#include <iostream>
using namespace std;

int main()

{
    int a;
    int b;

    a = 10;
    b = 5;

}

```

STEP 3 You can build and run the code but it won’t do much, other than store the values 10 and 5 to the integers a and b. To output the contents of the variables, add:

```
cout << a;
cout << ”\n”;
cout << b;
```

The cout << “\n”; part simply places a new line between the output of 10 and 5.

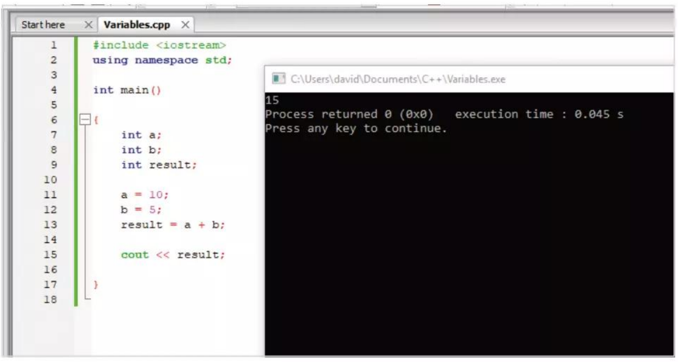


```
#include <iostream>
using namespace std;

int main()

{
    int a;
    int b;

    a = 10;
    b = 5;

    cout << a;
    cout << "\n";
    cout << b;

}

```

STEP 4 Naturally you can declare a new variable, call it result and output some simple arithmetic:

```
int result;

result = a + b;
cout << result;
```

Insert the above into the code as per the screenshot.



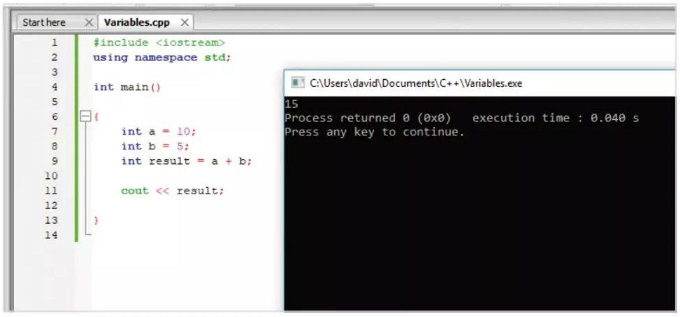
```
#include <iostream>
using namespace std;

int main()

{
    int a;
    int b;
    int result;

    a = 10;
    b = 5;
    result = a + b;

    cout << result;

}

```

```
15
Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

STEP 5

You can assign a value to a variable as soon as you declare it. The code you've typed in could look like this, instead:

```
int a = 10;
int b = 5;
int result = a + b;

cout << result;
```



```
#include <iostream>
using namespace std;

int main()

{

    int a = 10;
    int b = 5;
    int result = a + b;

    cout << result;

}
```

```
15
Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

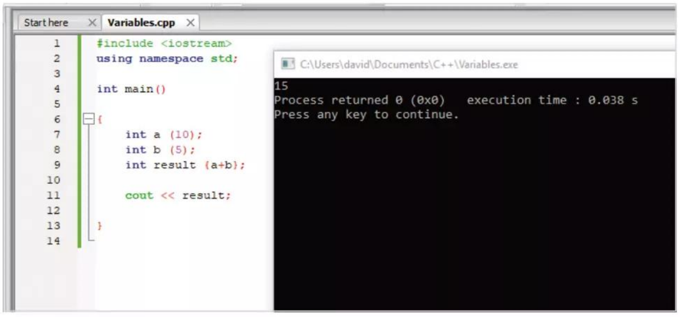
STEP 6

Specific to C++, you can also use the following to assign values to a variable as soon as you declare them:

```
int a (10);
int b (5);
```

Then, from the C++ 2011 standard, using curly brackets:

```
int result {a+b};
```



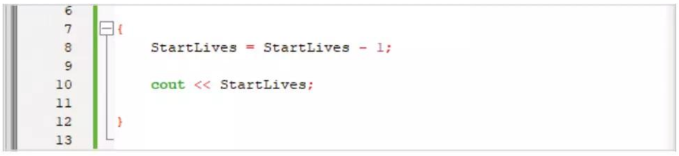
```
#include <iostream>
using namespace std;

int main()

{

    int a (10);
    int b (5);
    int result {a+b};

    cout << result;

}
```

```
15
Process returned 0 (0x0)   execution time : 0.038 s
Press any key to continue.
```

STEP 7

You can create global variables, which are variables that are declared outside any function and used in any function within the entire code. What you've used so far are local variables: variables used inside the function. For example:

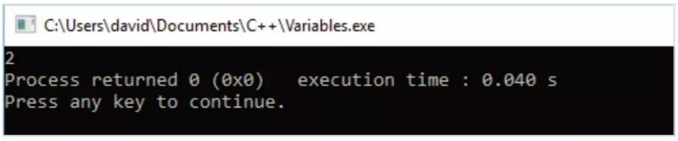
```
#include <iostream>
using namespace std;
int StartLives = 3;

int main ()

{
     startLives = StartLives - 1;
     cout << StartLives;
}
```



```
{

    StartLives = StartLives - 1;

    cout << StartLives;

}
```

STEP 8

The previous step creates the variable StartLives, which is a global variable. In a game, for example, a player's lives go up or down depending on how well or how bad they're doing. When the player restarts the game, the StartLives returns to its default state: 3. Here we've assigned 3 lives, then subtracted 1, leaving 2 lives left.

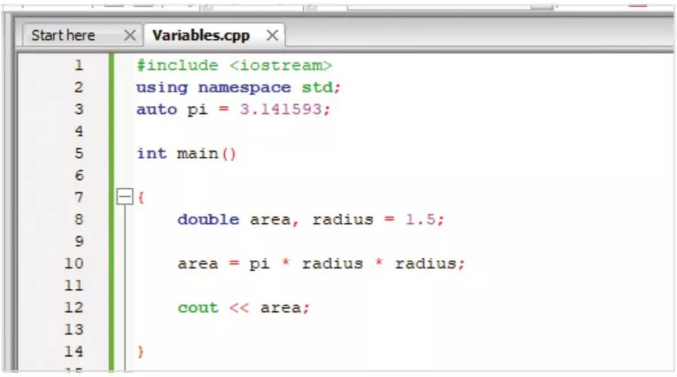


```
2
Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

STEP 9

The modern C++ compiler is far more intelligent than most programmers give it credit. While there are numerous data types you can declare for variables, you can in fact use the auto feature:

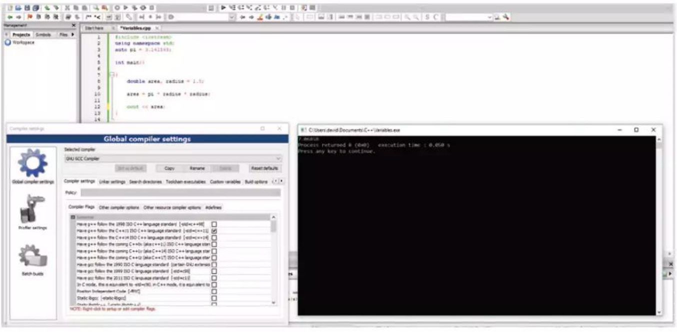
```
#include <iostream>
using namespace std;
auto pi = 3.141593;

int main()

{
    double area, radius = 1.5;

    area = pi * radius * radius;

    cout << area;

}
```



```
#include <iostream>
using namespace std;
auto pi = 3.141593;

int main()

{

    double area, radius = 1.5;

    area = pi * radius * radius;

    cout << area;

}
```

STEP 10

A couple of new elements here: first, auto won't work unless you go to Settings > Compiler and tick the box labelled 'Have G++ follow the C++11 ISO C++ Language Standard [-std=c++1]'. Then, the new data type, double, which means double-precision floating point value. Enable C++11, then build and run the code. The result should be 7.06858.



Data Types

Variables, as we've seen, store information that the programmer can then later call up, and manipulate if required. Variables are simply reserved memory locations that store the values the programmer assigns, depending on the data type used.

THE VALUE OF DATA

There are many different data types available for the programmer in C++, such as an integer, floating point, Boolean, character and so on. It's widely accepted that there are seven basic data types, often called Primitive Built-in Types; however, you can create your own data types should the need ever arise within your code.

The seven basic data types are:

TYPE	COMMAND
Integer	`Integer`
Floating Point	`float`
Character	`char`
Boolean	`bool`
Double Floating Point	`double`
Wide Character	`wchar_t`
No Value	`void`

These basic types can also be extended using the following modifiers: Long, Short, Signed and Unsigned. Basically this means the modifiers can expand the minimum and maximum range values for each data type. For example, the int data type has a default value range of -2147483648 to 2147483647, a fair value, you would agree.

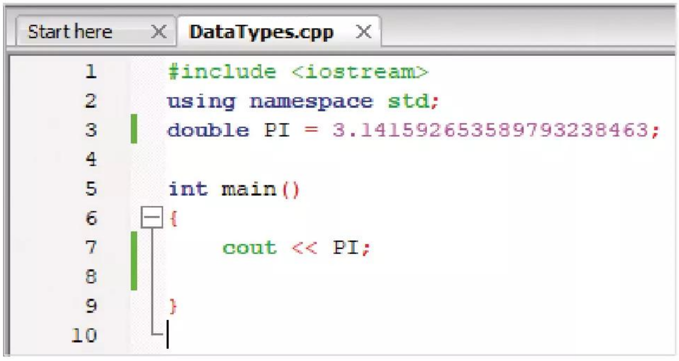
Now, if you were to use one of the modifiers, the range alters:

Unsigned int = 0 to 4294967295
Signed int = -2147483648 to 2147483647
Short int = -32768 to 32767
Unsigned Short int = 0 to 65,535
Signed Short int = -32768 to 32767
Long int = -2147483647 to 2147483647
Signed Long int = -2147483647 to 2147483647
Unsigned Long int = 0 to 4294967295

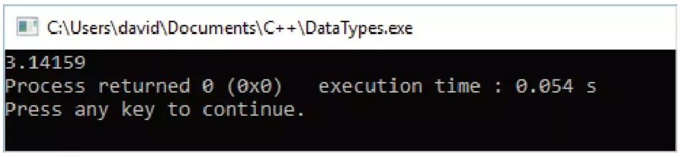
Naturally you can get away with using the basic type without the modifier, as there's plenty of range provided with each data type. However, it's considered good C++ programming practise to use the modifiers when possible.

There are issues when using the modifiers though. Double represents a double-floating point value, which you can use for incredibly accurate numbers but those numbers are only accurate up to the fifteenth decimal place. There's also the problem when displaying such numbers in C++ using the cout function, in that cout by default only outputs the first five decimal places. You can combat that by adding a cout.precision () function and adding a value inside the brackets, but even then you're still limited by the accuracy of the double data type. For example, try this code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout << PI;
}
```



```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout << PI;

}
```

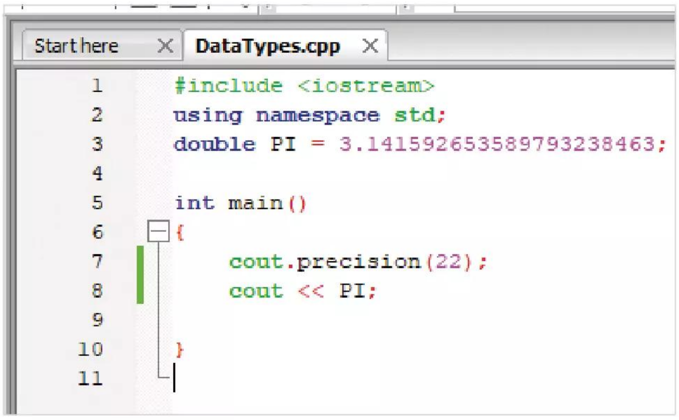


```
3.14159
Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.
```

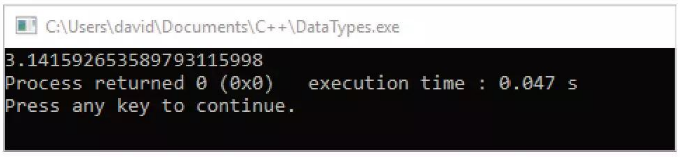
Build and run the code and as you can see the output is only 3.14159, representing cout's limitations in this example.

You can alter the code including the aforementioned cout.precision function, for greater accuracy. Take precision all the way up to 22 decimal places, with the following code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout.precision(22);
    cout << PI;

}
```



```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout.precision(22);
    cout << PI;

}
```



```
3.141592653589793115998
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Again, build and run the code; as you can see from the command line window, the number represented by the variable PI is different to the number you've told C++ to use in the variable. The output reads the value of PI as 3.141592653589793115998, with the numbers going awry from the fifteenth decimal place.

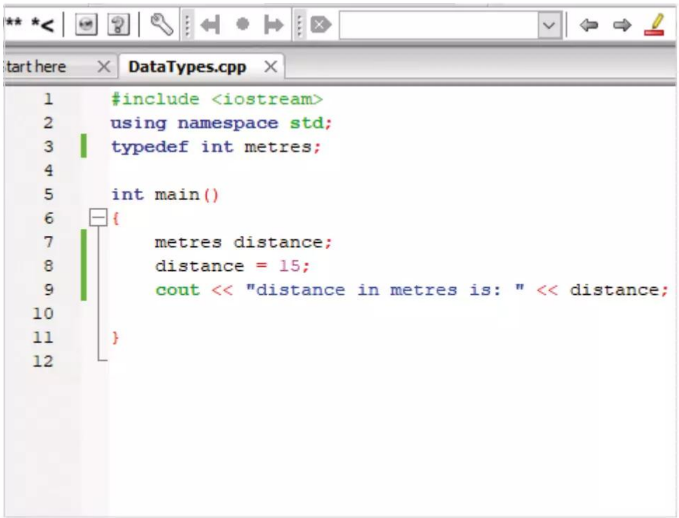
Calculator
Scientific
DEG
HYP
F-E
MC
MR
M+
M-
MS
M˅
x²
xʸ
sin
cos
tan
√
10ˣ
log
Exp
Mod
↑
CE
C
⌫
÷
π
7
8
9
×
n!
4
5
6
−
±
1
2
3
+
(
)
0
.
=

15.142857142857142857142857142857

This is mainly due to the conversion from binary in the compiler and that the IEEE 754 double precision standard occupies 64-bits of data, of which 52-bits are dedicated to the significant (the significant digits in a floating-point number) and roughly 3.5-bits are taken holding the values 0 to 9. If you divide 53 by 3.5, then you arrive at 15.142857 recurring, which is 15-digits of precision.

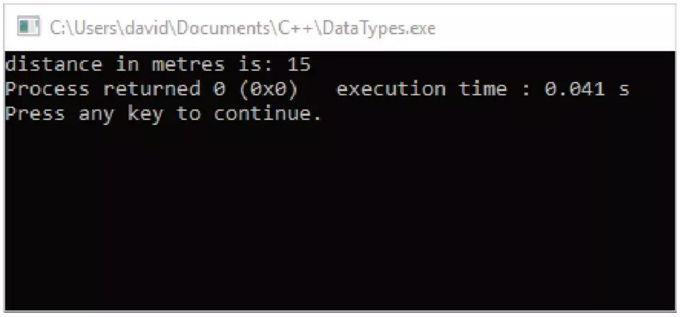
To be honest, if you're creating code that needs to be accurate to more than fifteen decimal places, then you wouldn't be using C++, you would use some scientific specific language with C++ as the connective tissue between the two languages.

You can create your own data types, using an alias-like system called typedef. For example:



```
#include <iostream>
using namespace std;
typedef int metres;

int main()
{
    metres distance;
    distance = 15;
    cout << "distance in metres is: " << distance;

}
```

```
#include <iostream>
using namespace std;
typedef int metres;

int main()
{
    metres distance;
    distance = 15;
    cout << "distance in metres is: " << distance;

}
```



```
distance in metres is: 15
Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

This code when executed creates a new int data type called metres. Then, in the main code block, there's a new variable called distance, which is an integer; so you're basically telling the compiler that there's another name for int. We assigned the value 15 to distance and displayed the output: distance in metres is 15.

It might sound a little confusing to begin with but the more you use C++ and create your own code, the easier it becomes.

Strings

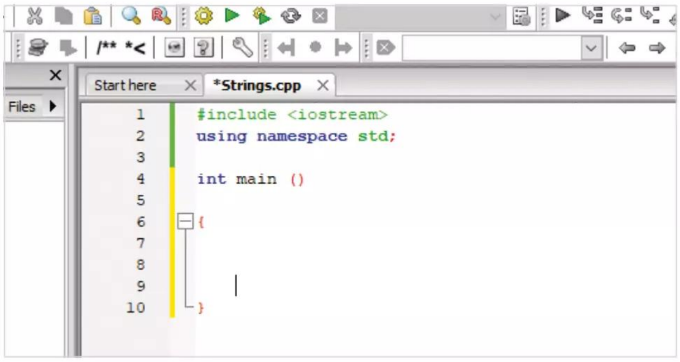
Strings are objects that represent and hold sequences of characters. For example, you could have a universal greeting in your code ‘Welcome’ and assign that as a string to be called up wherever you like in the program.

STRING THEORY

There are different ways in which you can create a string of characters, which historically are all carried over from the original C language, and are still supported by C++.

STEP 1 To create a string you use the char function. Open a new C++ file and begin with the usual header:

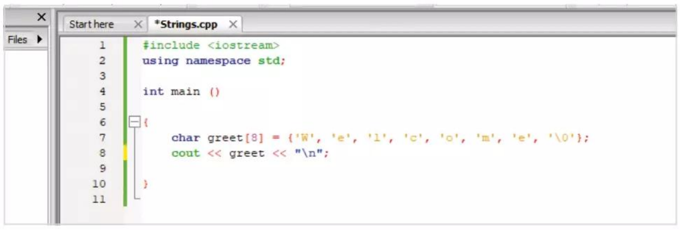
```
#include <iostream>
using namespace std;

int main ()

{

}
```



```
#include <iostream>
using namespace std;

int main ()

{



}
```

STEP 2 It’s easy to confuse a string with an array. Here’s an array, which can be terminated with a null character:

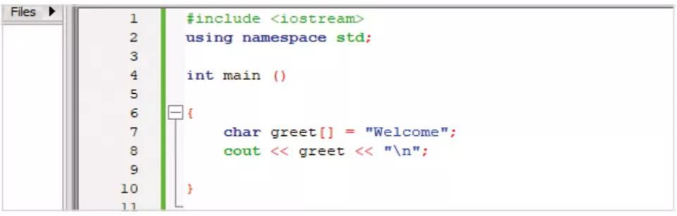
```
#include <iostream>
using namespace std;

int main ()

{
    char greet[8] = {‘W’, ‘e’, ‘l’, ‘c’, ‘o’, ‘m’,
‘e’, ‘\0’};
    cout << greet << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{
    char greet[8] = {'W', 'e', 'l', 'c', 'o', 'm', 'e', '\0'};
    cout << greet << "\n";

}
```

STEP 3 Build and run the code, and ‘Welcome’ appears on the screen. While this is perfectly fine, it’s not a string. A string is a class, which defines objects that can be represented as a stream of characters and doesn’t need to be terminated like an array. The code can therefore be represented as:

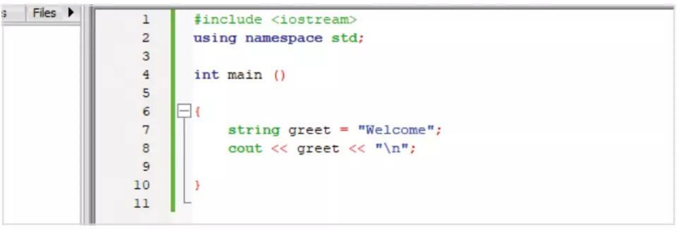
```
#include <iostream>
using namespace std;

int main ()

{
    char greet[] = “Welcome”;
    cout << greet << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{
    char greet[] = "Welcome";
    cout << greet << "\n";

}
```

STEP 4 In C++ there’s also a string function, which works in much the same way. Using the greeting code again, you can enter:

```
#include <iostream>
using namespace std;

int main ()

{
    string greet = “Welcome”;
    cout << greet << “\n”;

}
```

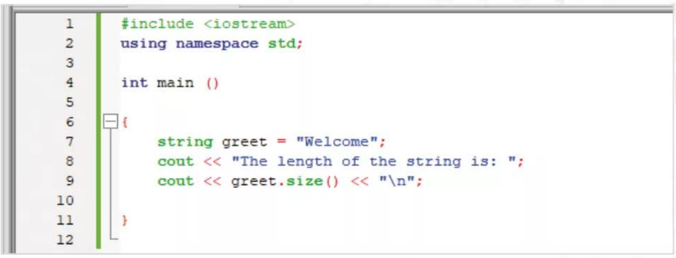


```
#include <iostream>
using namespace std;

int main ()

{
    string greet = "Welcome";
    cout << greet << "\n";

}
```

STEP 5

There are also many different operations that you can apply with the string function. For instance, to get the length of a string you can use:

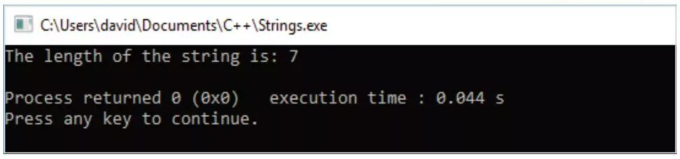
```
#include <iostream>
using namespace std;

int main ()

{
    string greet = “Welcome”;
    cout << “The length of the string is: ”;
    cout << greet.size() << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{
    string greet = "Welcome";
    cout << "The length of the string is: ";
    cout << greet.size() << "\n";

}
```

STEP 6

You can see that we used greet.size() to output the length, the number of characters there are, of the contents of the string. Naturally, if you call your string something other than greet, then you need to change the command to reflect this. It's always stringname.operation. Build and run the code to see the results.

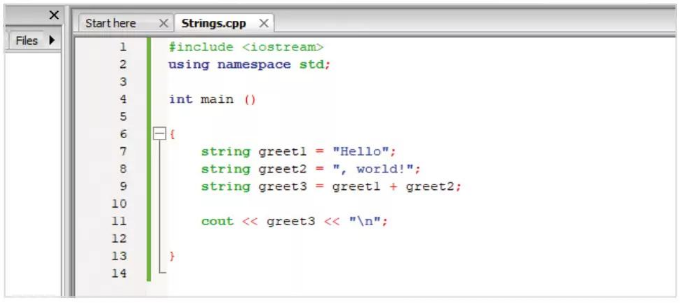


```
The length of the string is: 7

Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
```

STEP 7

You can of course add strings together, or rather combine them to form longer strings:

```
#include <iostream>
using namespace std;

int main ()

{
    string greet1 = “Hello”;
    string greet2 = “, world!”;
    string greet3 = greet1 + greet2;

    cout << greet3 << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{
    string greet1 = "Hello";
    string greet2 = ", world!";
    string greet3 = greet1 + greet2;

    cout << greet3 << "\n";

}
```

STEP 8

Just as you might expect, you can mix in an integer and store something to do with the string. In this example, we created int length, which stores the result of string.size() and outputs it to the user:

```
#include <iostream>
using namespace std;

int main ()

{
    int length;
    string greet1 = “Hello”;
    string greet2 = “, world!”;
    string greet3 = greet1 + greet2;

    length = greet3.size();

    cout << “The length of the combined strings
is: ” << length << “\n”;

}
```

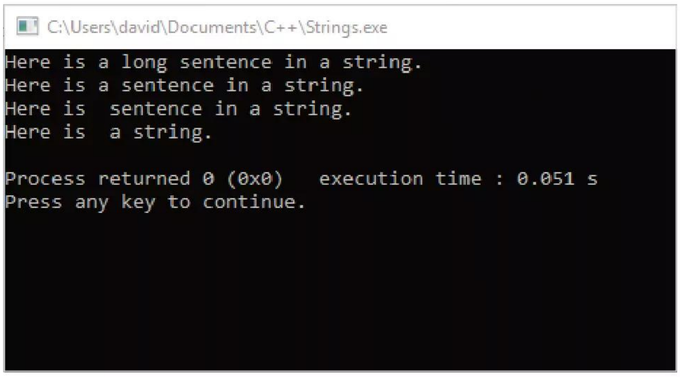
STEP 9

Using the available operations that come with the string function, you can manipulate the contents of a string. For example, to remove characters from a string you could use:

```
#include <iostream>
using namespace std;

int main ()

{
  string strg (“Here is a long sentence in a
string.”);
  cout << strg << ‘\n’;

  strg.erase (10,5);
  cout << strg << ‘\n’;

  strg.erase (strg.begin()+8);
  cout << strg << ‘\n’;

  strg.erase (strg.begin()+9, strg.end()-9);
  cout << strg << ‘\n’;

}
```

STEP 10

It's worth spending some time playing around with the numbers, which are the character positions in the string. Occasionally, it can be hit and miss whether you get it right, so practice makes perfect. Take a look at the screenshot to see the result of the code.



```
Here is a long sentence in a string.
Here is a sentence in a string.
Here is  sentence in a string.
Here is  a string.

Process returned 0 (0x0)   execution time : 0.051 s
Press any key to continue.
```

C++ Maths

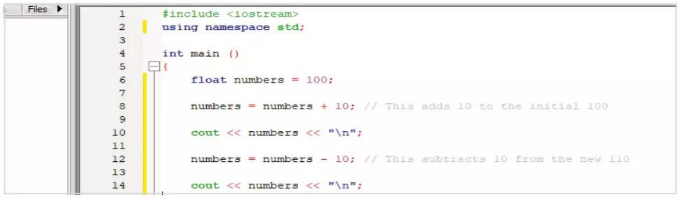
Programming is mathematical in nature and as you might expect, there’s plenty of built-in scope for some quite intense maths. C++ has a lot to offer someone who’s implementing mathematical models into their code. It can be extremely complex or relatively simple.

C++ = MC²

The basic mathematical symbols apply in C++ as they do in most other programming languages. However, by using the C++ Math Library, you can also calculate square roots, powers, trig and more.

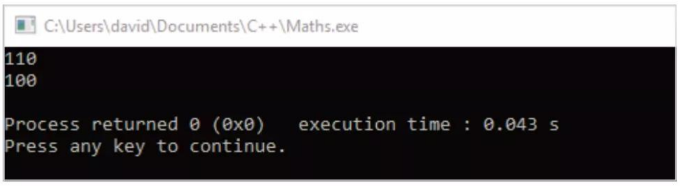
STEP 1 C++’s mathematical operations follow the same patterns as those taught in school, in that multiplication and division take precedence over addition and subtraction. You can alter that though. For now, create a new file and enter:

```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers + 10; // This adds 10 to the
initial 100

    cout << numbers << “\n”;

    numbers = numbers - 10; // This subtracts 10
from the new 110

    cout << numbers << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers + 10; // This adds 10 to the initial 100

    cout << numbers << "\n";

    numbers = numbers - 10; // This subtracts 10 from the new 110

    cout << numbers << "\n";
```

STEP 2 While simple, it does get the old maths muscle warmed up. Note that we used a float for the numbers variable. While you can happily use an integer, if you suddenly started to use decimals, you would need to change to a float or a double, depending on the accuracy needed. Run the code and see the results.

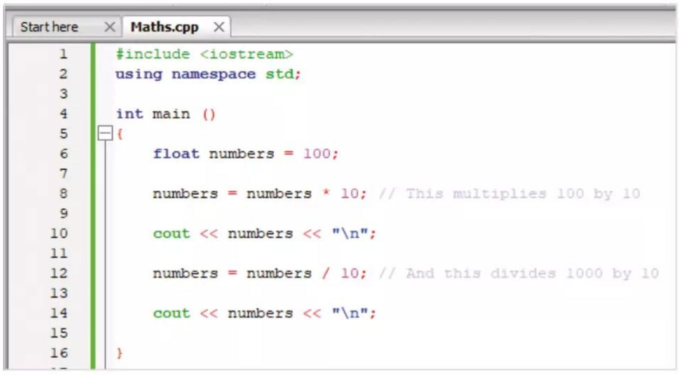


C:\Users\david\Documents\C++\Maths.exe

```
110
100

Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

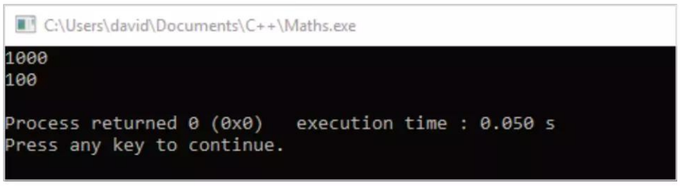
STEP 3 Multiplication and division can be applied as such:

```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers * 10; // This multiplies 100
by 10

    cout << numbers << “\n”;

    numbers = numbers / 10; // And this divides
1000 by 10

    cout << numbers << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers * 10; // This multiplies 100 by 10

    cout << numbers << "\n";

    numbers = numbers / 10; // And this divides 1000 by 10

    cout << numbers << "\n";

}
```

STEP 4 Again, execute the simple code and see the results. While not particularly interesting, it’s a start into C++ maths. We used a float here, so you can play around with the code and multiply by decimal places, as well as divide, add and subtract.



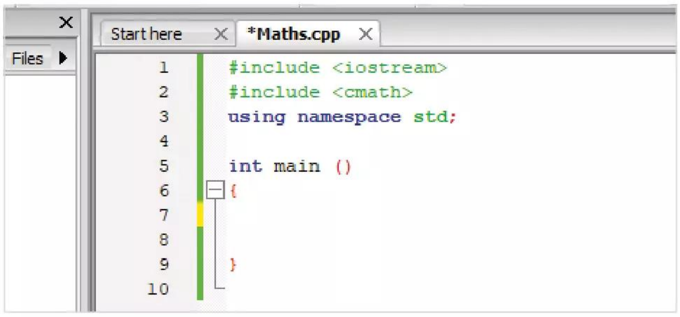
C:\Users\david\Documents\C++\Maths.exe

```
1000
100

Process returned 0 (0x0)   execution time : 0.050 s
Press any key to continue.
```

STEP 5

The interesting maths content comes when you call upon the C++ Math Library. Within this header are dozens of mathematical functions along with further operations. Everything from computing cosine to arc tangent with two parameters, to the value of PI. You can call the header with:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{

}
```

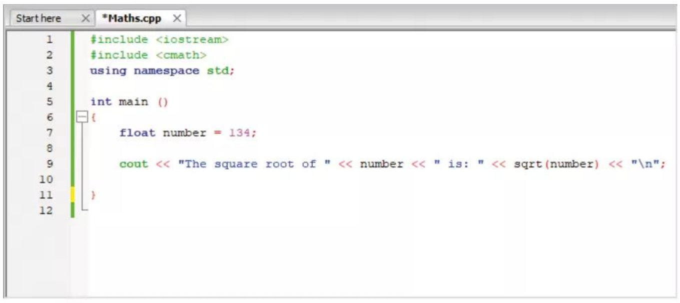


```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{


}
```

STEP 6

Start by getting the square root of a number:

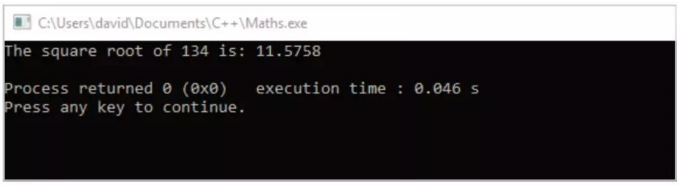
```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 134;

    cout << “The square root of ” << number << ”
is: ” << sqrt(number) << “\n”;

}
```



```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 134;

    cout << "The square root of " << number << " is: " << sqrt(number) << "\n";
}
```

STEP 7

Here we created a new float called number and used the sqrt(number) function to display the square root of 134, the value of the variable, number. Build and run the code, and your answer reads 11.5758.

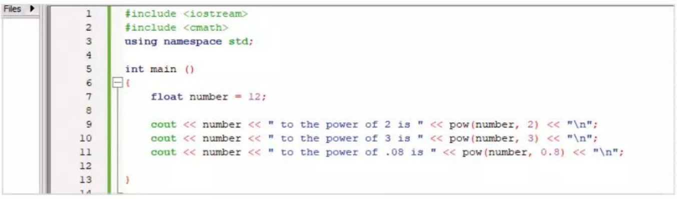


```
The square root of 134 is: 11.5758

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

STEP 8

Calculating powers of numbers can be done with:

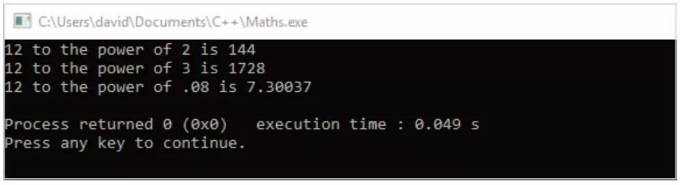
```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 12;

    cout << number << ” to the power of 2 is ” <<
pow(number, 2) << “\n”;
    cout << number << ” to the power of 3 is ” <<
pow(number, 3) << “\n”;
    cout << number << ” to the power of .08 is ”
<< pow(number, 0.8) << “\n”;

}
```



```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 12;

    cout << number << " to the power of 2 is " << pow(number, 2) << "\n";
    cout << number << " to the power of 3 is " << pow(number, 3) << "\n";
    cout << number << " to the power of .08 is " << pow(number, 0.8) << "\n";

}
```

STEP 9

Here we created a float called number with the value of 12, and the pow(variable, power) is where the calculation happens. Of course, you can calculate powers and square roots without using variables. For example, pow (12, 2) outputs the same value as the first cout line in the code.

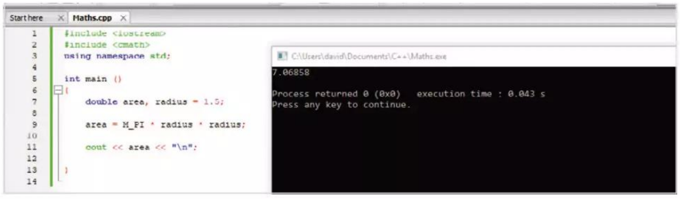


```
12 to the power of 2 is 144
12 to the power of 3 is 1728
12 to the power of .08 is 7.30037

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

STEP 10

The value of Pi is also stored in the cmath header library. It can be called up with the M_PI function. Enter cout << M_PI; into the code and you get 3.14159; or you can use it to calculate:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    double area, radius = 1.5;

    area = M_PI * radius * radius;

    cout << area << “\n”;

}
```



```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    double area, radius = 1.5;

    area = M_PI * radius * radius;

    cout << area << "\n";

}
```

```
7.06858

Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

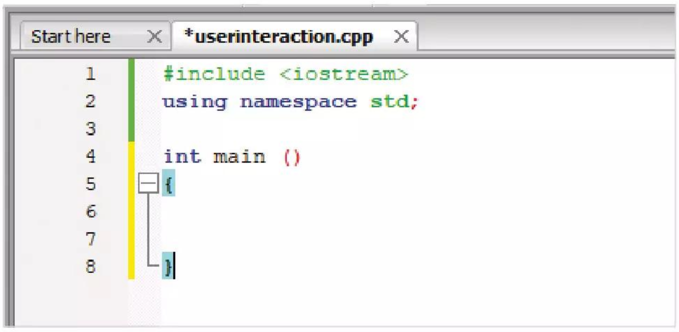
User Interaction

There’s nothing quite as satisfying as creating a program that responds to you. This basic user interaction is one of the most taught aspects of any language and with it you’re able to do much more than simply greet the user by name.

HELLO, DAVE

You have already used cout, the standard output stream, throughout our code. Now you’re going to be using cin, the standard input stream, to prompt a user response.

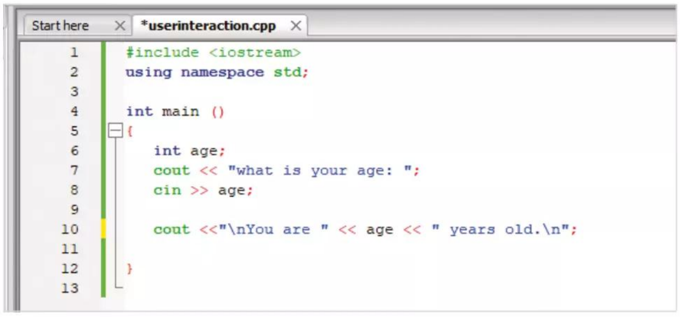
STEP 1 Anything that you want the user to input into the program needs to be stored somewhere in the system memory, so it can be retrieved and used. Therefore, any input must first be declared as a variable, so it’s ready to be used by the user. Start by creating a blank C++ file with headers.



```
#include <iostream>
using namespace std;

int main ()
{


}
```

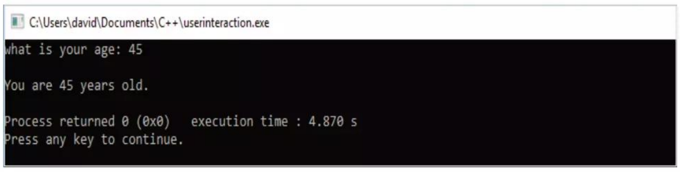
STEP 2 The data type of the variable must also match the type of input you want from the user. For example, to ask a user their age, you would use an integer like this:

```
#include <iostream>
using namespace std;

int main ()
{
   int age;
   cout << “what is your age: “;
   cin >> age;

   cout <<”\nYou are “ << age << “ years old.\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    int age;
    cout << "what is your age: ";
    cin >> age;

    cout <<"\nYou are " << age << " years old.\n";

}
```

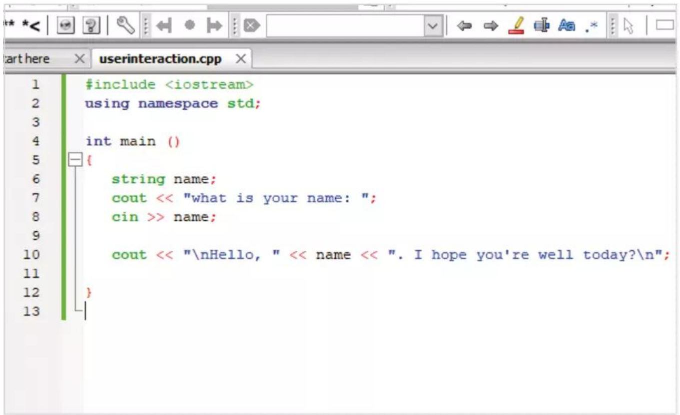
STEP 3 The cin command works in the opposite way from the cout command. With the first cout line you’re outputting ‘What is your age’ to the screen, as indicated with the chevrons. Cin uses opposite facing chevrons, indicating an input. The input is put into the integer age and called up in the second cout command. Build and run the code.



```
what is your age: 45

You are 45 years old.

Process returned 0 (0x0)   execution time : 4.870 s
Press any key to continue.
```

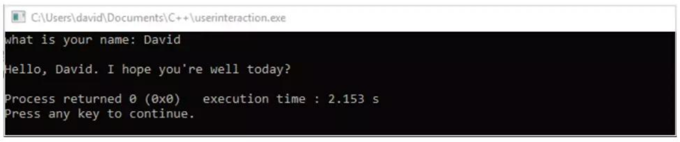
STEP 4 If you’re asking a question, you need to store the input as a string; to ask the user their name, you would use:

```
#include <iostream>
using namespace std;

int main ()
{
   string name;
   cout << “what is your name: “;
   cin >> name;

   cout << “\nHello, “ << name << “. I hope you’re
well today?\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    string name;
    cout << "what is your name: ";
    cin >> name;

    cout << "\nHello, " << name << ". I hope you're well today?\n";

}
```

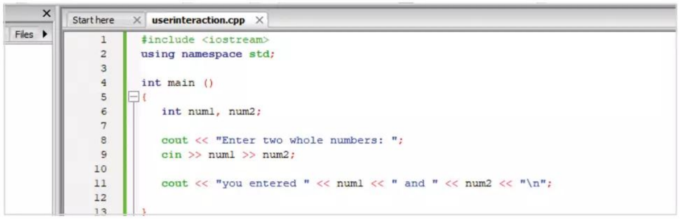
STEP 5 The principal works the same as the previous code. The user's input, their name, is stored in a string, because it contains multiple characters, and retrieved in the second cout line. As long as the variable 'name' doesn't change, then you can recall it wherever you like in your code.



```
what is your name: David

Hello, David. I hope you're well today?

Process returned 0 (0x0)   execution time : 2.153 s
Press any key to continue.
```

STEP 6 You can chain input requests to the user but just make sure you have a valid variable to store the input to begin with. Let's assume you want the user to enter two whole numbers:

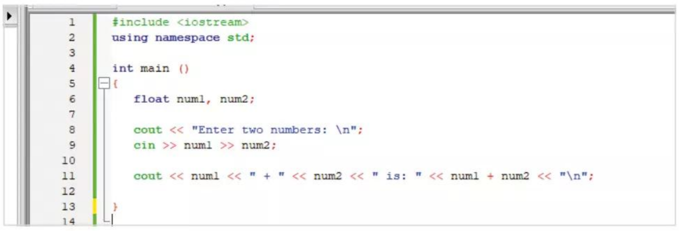
```
#include <iostream>
using namespace std;

int main ()
{
   int num1, num2;

   cout << “Enter two whole numbers: “;
   cin >> num1 >> num2;

   cout << “you entered “ << num1 << “ and “ <<
num2 << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
   int num1, num2;

   cout << "Enter two whole numbers: ";
   cin >> num1 >> num2;

   cout << "you entered " << num1 << " and " << num2 << "\n";

}
```

STEP 7 Likewise, inputted data can be manipulated once you have it stored in a variable. For instance, ask the user for two numbers and do some maths on them:

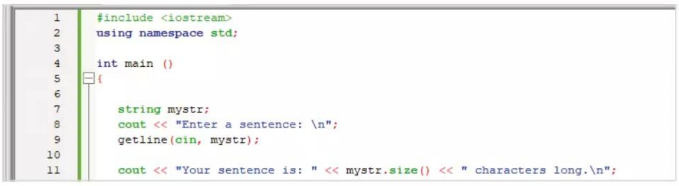
```
#include <iostream>
using namespace std;

int main ()
{
   float num1, num2;

   cout << “Enter two numbers: \n”;
   cin >> num1 >> num2;

   cout << num1 << “ + “ << num2 << “ is: “ <<
num1 + num2 << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
   float num1, num2;

   cout << "Enter two numbers: \n";
   cin >> num1 >> num2;

   cout << num1 << " + " << num2 << " is: " << num1 + num2 << "\n";

}
```

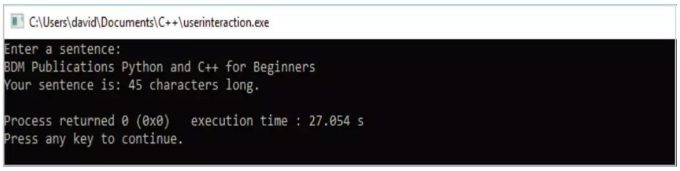
STEP 8 While cin works well for most input tasks, it does have a limitation. Cin always considers spaces as a terminator, so it's designed for just single words not multiple words. However, getline takes cin as the first argument and the variable as the second:

```
#include <iostream>
using namespace std;

int main ()
{

   string mystr;
   cout << “Enter a sentence: \n”;
   getline(cin, mystr);

   cout << “Your sentence is: “ << mystr.size() <<
“ characters long.\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{

   string mystr;
   cout << "Enter a sentence: \n";
   getline(cin, mystr);

   cout << "Your sentence is: " << mystr.size() << " characters long.\n";
```

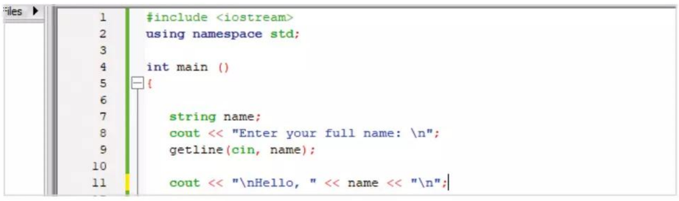
STEP 9 Build and execute the code, then enter a sentence with spaces. When you're done the code reads the number of characters. If you remove the getline line and replace it with cin >> mystr and try again, the result displays the number of characters up to the first space.



```
Enter a sentence:
BDM Publications Python and C++ for Beginners
Your sentence is: 45 characters long.

Process returned 0 (0x0)   execution time : 27.054 s
Press any key to continue.
```

STEP 10 Getline is usually a command that new C++ programmers forget to include. The terminating white space is annoying when you can't figure out why your code isn't working. In short, it's best to use getline(cin, variable) in future:

```
#include <iostream>
using namespace std;

int main ()
{

   string name;
   cout << “Enter your full name: \n”;
   getline(cin, name);

   cout << “\nHello, “ << name << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{

   string name;
   cout << "Enter your full name: \n";
   getline(cin, name);

   cout << "\nHello, " << name << "\n";
```

Python is one of the most respected and used programming languages in the world. It's designed to help beginners get into code, but also has some amazingly powerful features that are utilised by data scientists and engineers around the world.

You don't need to look too far to Python at work. It's used throughout the Internet, and helps drive some of the biggest projects and companies in a multitude of industries.

In this section, we will look at what you will need to get started with Python. First steps into a new learning experience are always the hardest, but we're here to help.

54 Why Python?
56 What Can You Do with Python?
58 Python in Numbers
60 Equipment You Will Need
62 Getting to Know Python
64 How to Set Up Python in Windows
66 How to Set Up Python in Linux
68 Python on the Pi

Introducing Python

Why Python?

There are many different programming languages available for the modern computer, and some still available for older 8 and 16-bit computers too. Some of these languages are designed for scientific work, others for mobile platforms and such. So why choose Python out of all the rest?

PYTHON POWER

Ever since the earliest home computers were available, enthusiasts, users and professionals have toiled away until the wee hours, slaving over an overheating heap of circuitry to create something akin to magic.

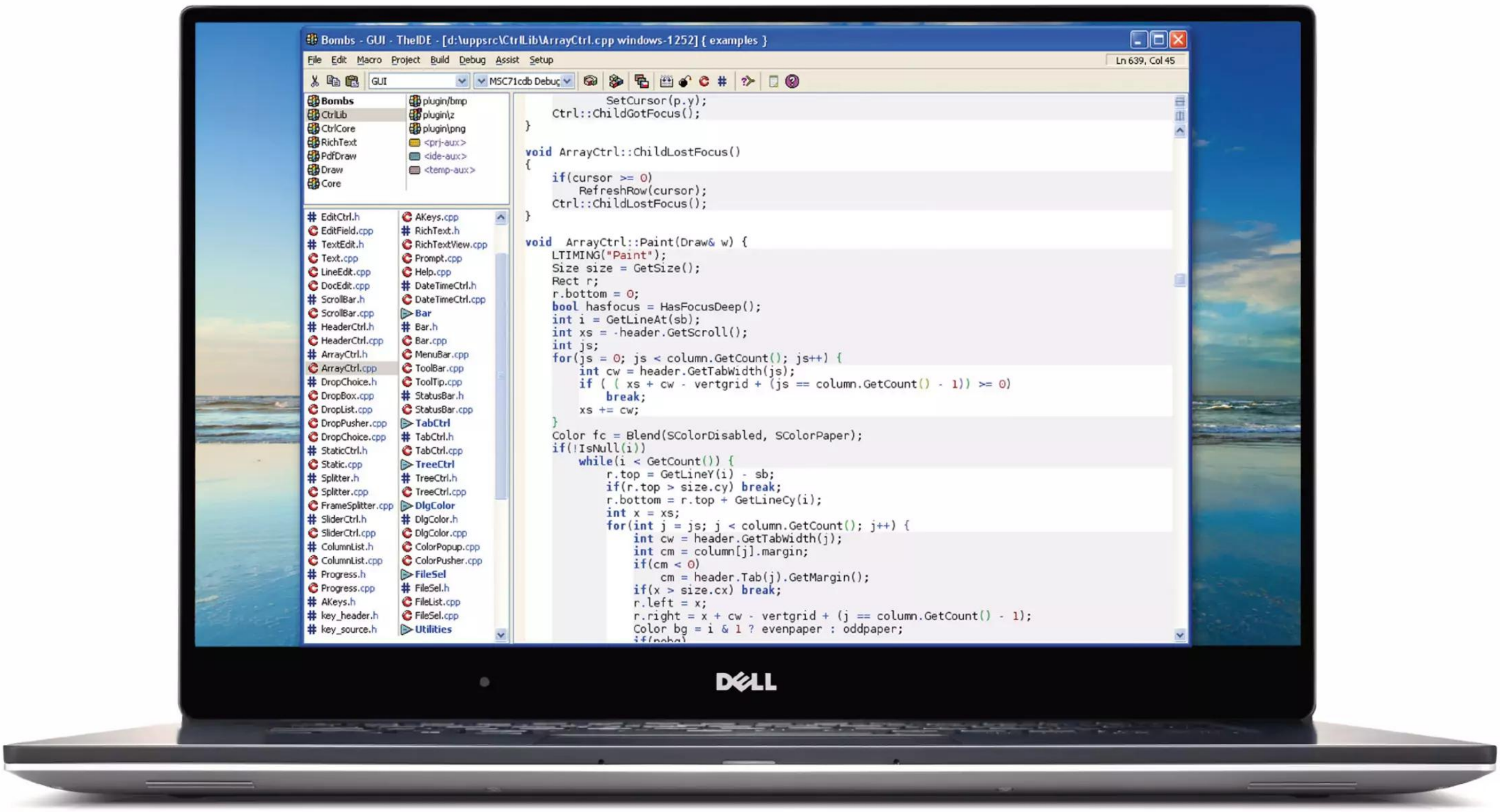
These pioneers of programming carved their way into a new frontier, forging small routines that enabled the letter 'A' to scroll across the screen. It may not sound terribly exciting to a generation that's used to ultra high-definition graphics and open world, multi-player online gaming. However, forty-something years ago it was blindingly brilliant.

Naturally these bedroom coders helped form the foundations for every piece of digital technology we use today. Some went on to become chief developers for top software companies, whereas others pushed the available hardware to its limits and founded the billion pound gaming empire that continually amazes us.

Regardless of whether you use an Android device, iOS device, PC, Mac, Linux, Smart TV, games console, MP3 player, GPS device built-in to a car, set-top box or a thousand other connected and 'smart' appliances, behind them all is programming.

All those aforementioned digital devices need instructions to tell them what to do, and allow them to be interacted with. These instructions form the programming core of the device and that core can be built using a variety of programming languages.

The languages in use today differ depending on the situation, the platform, the device's use and how the device will interact with its

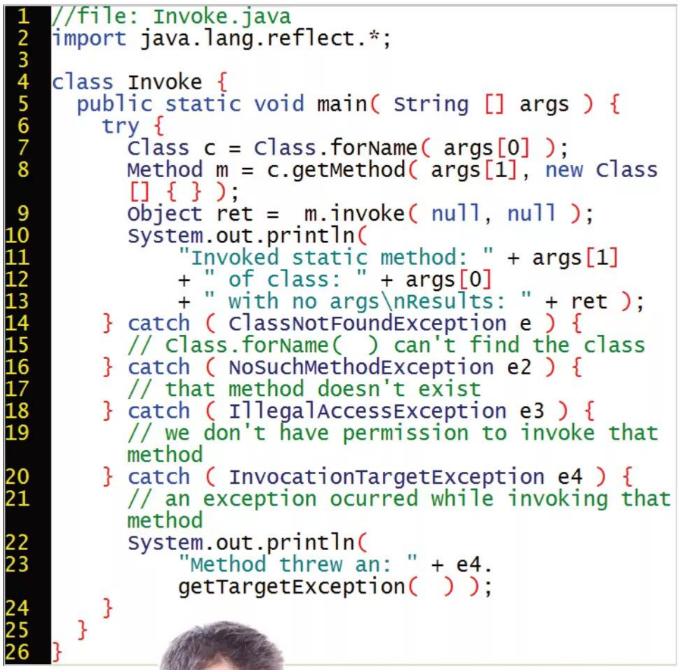

Bombs - GUI - TheIDE - [d:\uppsrc\CtrlLib\ArrayCtrl.cpp windows-1252] { examples }
File Edit Macro Project Build Debug Assist Setup
Ln 639, Col 45
GUI
MSC71cdb Debug
Bombs
CtrlLib
CtrlCore
RichText
PdfDraw
Draw
Core
plugin/bmp
plugin\z
plugin\png
<prj-aux>
<ide-aux>
<temp-aux>
EditCtrl.h
EditField.cpp
TextEdit.h
Text.cpp
LineEdit.cpp
DocEdit.cpp
ScrollBar.h
ScrollBar.cpp
HeaderCtrl.h
HeaderCtrl.cpp
ArrayCtrl.h
ArrayCtrl.cpp
DropChoice.h
DropBox.cpp
DropList.cpp
DropPusher.cpp
DropChoice.cpp
StaticCtrl.h
Static.cpp
Splitter.h
Splitter.cpp
FrameSplitter.cpp
SliderCtrl.h
SliderCtrl.cpp
ColumnList.h
ColumnList.cpp
Progress.h
Progress.cpp
AKeys.h
key_header.h
key_source.h
AKeys.cpp
RichText.h
RichTextView.cpp
Prompt.cpp
Help.cpp
DateTimeCtrl.h
DateTimeCtrl.cpp
Bar
Bar.h
Bar.cpp
MenuBar.cpp
ToolBar.cpp
ToolTip.cpp
StatusBar.h
StatusBar.cpp
TabCtrl
TabCtrl.h
TabCtrl.cpp
TreeCtrl
TreeCtrl.h
TreeCtrl.cpp
DlgColor
DlgColor.h
DlgColor.cpp
ColorPopup.cpp
ColorPusher.cpp
FileSel
FileSel.h
FileList.cpp
FileSel.cpp
Utilities
DELL

```
        SetCursor(p.y);
    Ctrl::ChildGotFocus();
}

void ArrayCtrl::ChildLostFocus()
{
    if(cursor >= 0)
        RefreshRow(cursor);
    Ctrl::ChildLostFocus();
}

void  ArrayCtrl::Paint(Draw& w) {
    LTIMING("Paint");
    Size size = GetSize();
    Rect r;
    r.bottom = 0;
    bool hasfocus = HasFocusDeep();
    int i = GetLineAt(sb);
    int xs = -header.GetScroll();
    int js;
    for(js = 0; js < column.GetCount(); js++) {
        int cw = header.GetTabWidth(js);
        if ( ( xs + cw - vertgrid + (js == column.GetCount() - 1)) >= 0)
            break;
        xs += cw;
    }
    Color fc = Blend(SColorDisabled, SColorPaper);
    if(!IsNull(i))
        while(i < GetCount()) {
            r.top = GetLineY(i) - sb;
            if(r.top > size.cy) break;
            r.bottom = r.top + GetLineCy(i);
            int x = xs;
            for(int j = js; j < column.GetCount(); j++) {
                int cw = header.GetTabWidth(j);
                int cm = column[j].margin;
                if(cm < 0)
                    cm = header.Tab(j).GetMargin();
                if(x > size.cx) break;
                r.left = x;
                r.right = x + cw - vertgrid + (j == column.GetCount() - 1);
                Color bg = i & 1 ? evenpaper : oddpaper;
```

environment or users. Operating systems, such as Windows, macOS and such are usually a combination of C++, C#, assembly and some form of visual-based language. Games generally use C++ whilst web pages can use a plethora of available languages such as HTML, Java, Python and so on.

More general-purpose programming is used to create programs, apps, software or whatever else you want to call them. They're widely used across all hardware platforms and suit virtually every conceivable application. Some operate faster than others and some are easier to learn and use than others. Python is one such general-purpose language.

Python is what's known as a High-Level Language, in that it 'talks' to the hardware and operating system using a variety of arrays, variables, objects, arithmetic, subroutines, loops and countless more interactions. Whilst it's not as streamlined as a Low-Level Language, which can deal directly with memory addresses, call stacks and registers, its benefit is that it's universally accessible and easy to learn.



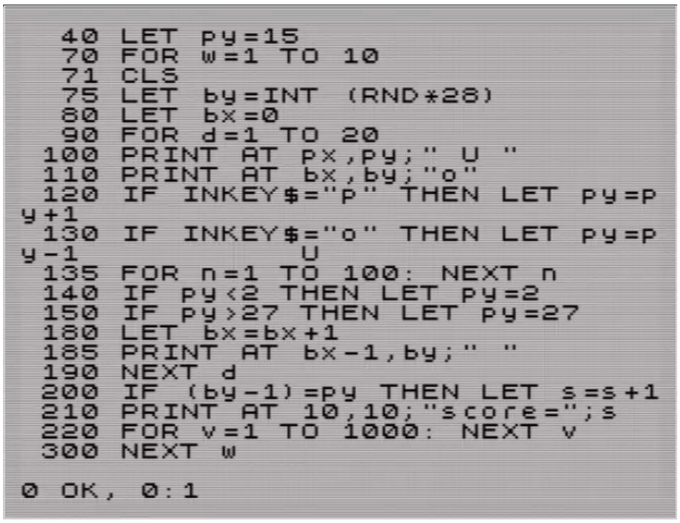
```
//file: Invoke.java
import java.lang.reflect.*;

class Invoke {
  public static void main( String [] args ) {
    try {
      Class c = Class.forName( args[0] );
      Method m = c.getMethod( args[1], new Class
      [] { } );
      Object ret =  m.invoke( null, null );
      System.out.println(
          "Invoked static method: " + args[1]
          + " of class: " + args[0]
          + " with no args\nResults: " + ret );
    } catch ( ClassNotFoundException e ) {
      // Class.forName(  ) can't find the class
    } catch ( NoSuchMethodException e2 ) {
      // that method doesn't exist
    } catch ( IllegalAccessException e3 ) {
      // we don't have permission to invoke that
      method
    } catch ( InvocationTargetException e4 ) {
      // an exception ocurred while invoking that
      method
      System.out.println(
          "Method threw an: " + e4.
          getTargetException(  ) );
    }
  }
}
```

Java is a powerful language that's used in web pages, set-top boxes, TVs and even cars.

Python was created over twenty six years ago and has evolved to become an ideal beginner's language for learning how to program a computer. It's perfect for the hobbyist, enthusiast, student, teacher and those who simply need to create their own unique interaction between either themselves or a piece of external hardware and the computer itself.

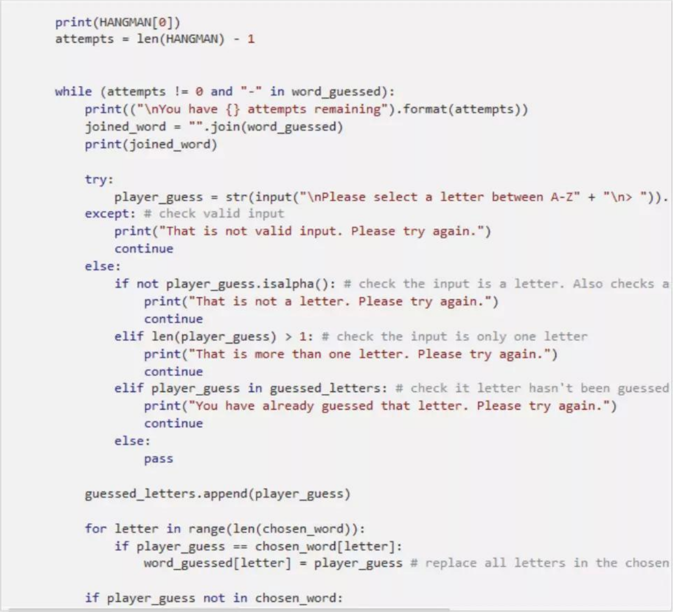
Python is free to download, install and use and is available for Linux, Windows, macOS, MS-DOS, OS/2, BeOS, IBM i-series machines, and even RISC OS. It has been voted one of the top five programming languages in the world and is continually evolving ahead of the hardware and Internet development curve.

So to answer the question: why Python? Simply put, it's free, easy to learn, exceptionally powerful, universally accepted, effective and a superb learning and educational tool.



```
  40 LET py=15
  70 FOR w=1 TO 10
  71 CLS
  75 LET by=INT (RND*28)
  80 LET bx=0
  90 FOR d=1 TO 20
 100 PRINT AT px,py;" U "
 110 PRINT AT bx,by;"o"
 120 IF INKEY$="p" THEN LET py=p
y+1
 130 IF INKEY$="o" THEN LET py=p
y-1          U
 135 FOR n=1 TO 100: NEXT n
 140 IF py<2 THEN LET py=2
 150 IF py>27 THEN LET py=27
 180 LET bx=bx+1
 185 PRINT AT bx-1,by;" "
 190 NEXT d
 200 IF (by-1)=py THEN LET s=s+1
 210 PRINT AT 10,10;"score=";s
 220 FOR v=1 TO 1000: NEXT v
 300 NEXT w

0 OK, 0:1
```

BASIC was once the starter language that early 8-bit home computer users learned.



```
print(HANGMAN[0])
attempts = len(HANGMAN) - 1


while (attempts != 0 and "-" in word_guessed):
    print(("\nYou have {} attempts remaining").format(attempts))
    joined_word = "".join(word_guessed)
    print(joined_word)

    try:
        player_guess = str(input("\nPlease select a letter between A-Z" + "\n> ")).
    except: # check valid input
        print("That is not valid input. Please try again.")
        continue
    else:
        if not player_guess.isalpha(): # check the input is a letter. Also checks a
            print("That is not a letter. Please try again.")
            continue
        elif len(player_guess) > 1: # check the input is only one letter
            print("That is more than one letter. Please try again.")
            continue
        elif player_guess in guessed_letters: # check it letter hasn't been guessed
            print("You have already guessed that letter. Please try again.")
            continue
        else:
            pass

    guessed_letters.append(player_guess)

    for letter in range(len(chosen_word)):
        if player_guess == chosen_word[letter]:
            word_guessed[letter] = player_guess # replace all letters in the chosen

    if player_guess not in chosen_word:
```

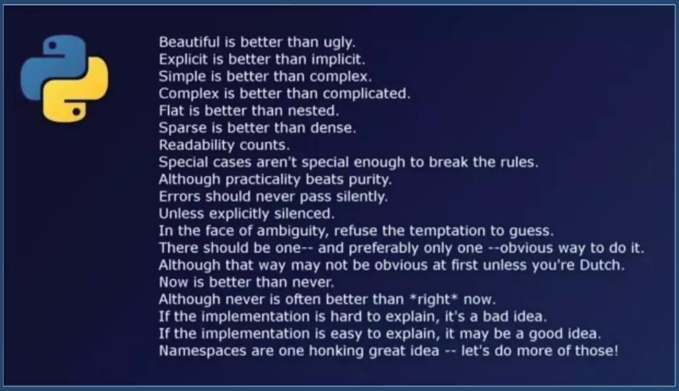
Python is a more modern take on BASIC, it's easy to learn and makes for an ideal beginner's programming language.

What Can You Do with Python?

Python is an open-source, object-oriented programming language that's simple to understand and write with, yet also powerful and extremely malleable. It's these characteristics that help make it such an important language to learn.

Python's ability to create highly readable code, within a small set of instructions, has a considerable impact on our modern digital world. From being an ideal first programmers' choice through to being able to create interactive stories and games; from scientific applications through to artificial Intelligence and web-based applications, the only limit to Python is the imagination of the person coding in it.

It's Python's malleable design that makes it an ideal language for many different situations and roles. Even certain aspects of the coding world that require more efficient code still use Python. For example, NASA utilises Python both as a stand-alone language and as a bridge between other programming languages. This way, NASA scientists and engineers are able to get to the data they need without having to cross multiple language barriers; Python fills the gaps and provides the means to get the job done. You'll find lots of examples of this, where Python is acting behind the scenes. This is why it's such an important language to learn.



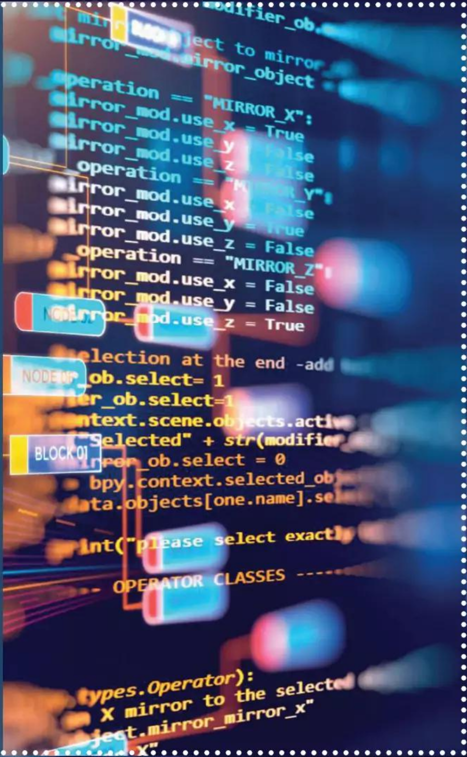
```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

BIG DATA

Big data is a buzzword you're likely to have come across in the last couple of years. Basically, it means extremely large data sets that are available for analysis to reveal patterns, trends and interactions between humans, society and technology. Of course, it's not just limited to those areas, big data is currently being used in a variety of industries, from social media to health and welfare, engineering to space exploration and beyond.

Python plays a substantial role in the world of big data. It's extensively used to analyse huge chunks of the available big data and extract specific information based on what the user/company requires from the wealth of numbers present. Thanks to an impressive set of data processing libraries, Python makes the act of getting to the data, in amongst the numbers, that counts and presenting it in a fashion that's readable and useable for humans.

There are countless libraries and freely available modules that enable fast, secure and more importantly, accurate processing of data from the likes of supercomputing clusters. For example, CERN uses a custom Python module to help analyse the 600 million collisions per second that the Large Hadron Collider (LHC) produces. A different language handles the raw data, but Python is present to help sift through the data so scientists can get to the content they want without the need to learn a far more complex programming language.



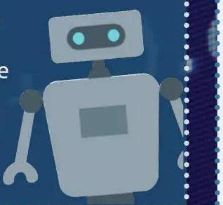
ARTIFICIAL INTELLIGENCE

Artificial Intelligence and Machine Learning are two of the most groundbreaking aspects of modern computing. AI is the umbrella term used for any computing process wherein the machine is doing something intelligent, working and reacting in similar ways to humans. Machine Learning is a subset of AI and provides the overall AI system with the ability to learn from its experiences.

However, AI isn't simply the creation of autonomous robots intent on wiping out human civilisation. Indeed, AI can be found in a variety of day-to-day computing applications where the 'machine', or more accurately the code, needs to learn from the actions of some form of input and anticipate what the input is likely to require, or do, next.

This model can be applied to Facebook, Google, Twitter, Instagram and so on. Have you ever looked up a celebrity on Instagram and then discovered that your searches within other social media platforms are now specifically targeted toward similar celebrities? This is a prime example of using AI in targeted advertising and behind the code and algorithms that predict what you're looking for, is Python.

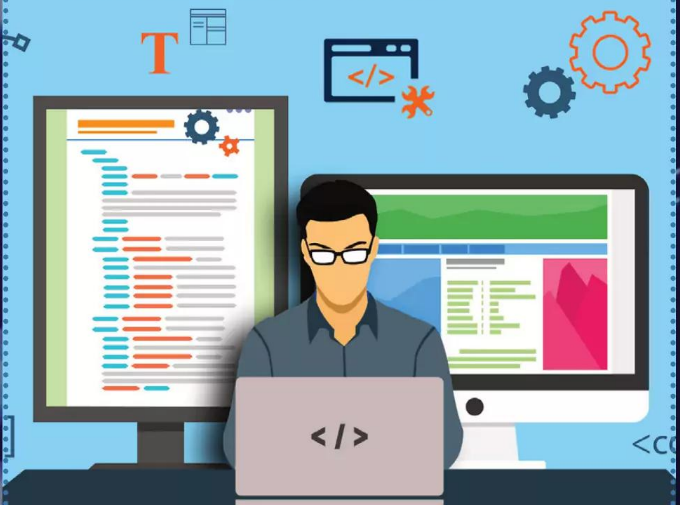
Spotify, for example, uses Python based code, among other things, to analyse your musical habits and offer playlists based on what you've listened to in the past. It's all clever stuff and moving forward, Python is at the forefront of the way the Internet will work in the future.

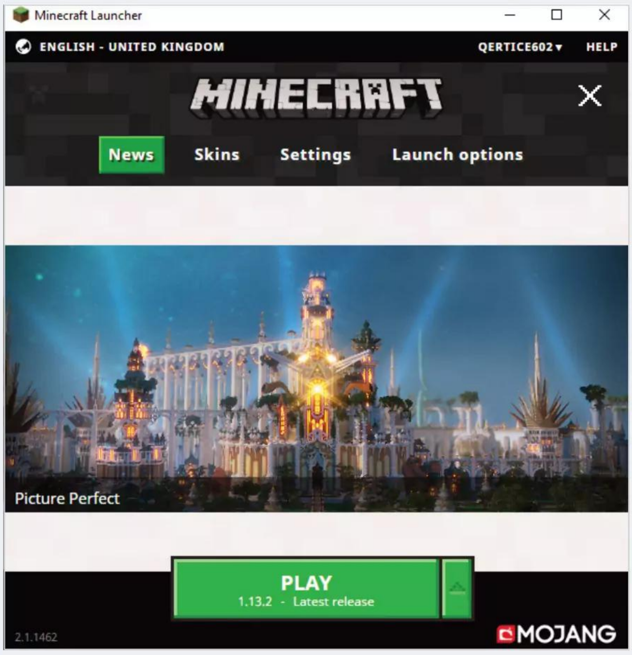


WEB DEVELOPMENT

Web development has moved on considerably since the early days of HTML scripting in a limited text editor. The many frameworks and web management services available now means that building a page has become increasingly complex.

With Python, the web developer has the ability to create dynamic and highly secure web apps, enabling interaction with other web services and apps such as Instagram and Pinterest. Python also allows the collection of data from other websites and even apps built within other websites.





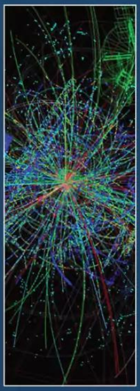
GAMING

Although you won't find too many triple-A rated games coded using Python, you may be surprised to learn that Python is used as an extra on many of the high-ranking modern games.

The main use of Python in gaming comes in the form of scripting, where a Python script can add customisations to the core game engine. Many map editors are Python compatible and you will also come across it if you build any mods for games, such as The Sims.

A lot of the online, MMORPG (Massively Multiplayer Online Role-Playing Game) games available utilise Python as a companion language for the server-side elements. These include: code to search for potential cheating, load balancing across the game's servers, player skill matchmaking and to check whether the player's client-side game matches the server's versions. There's also a Python module that can be included in a Minecraft server, enabling the server admin to add blocks, send messages and automate a lot of the background complexities of the game.

PYTHON EVERYWHERE

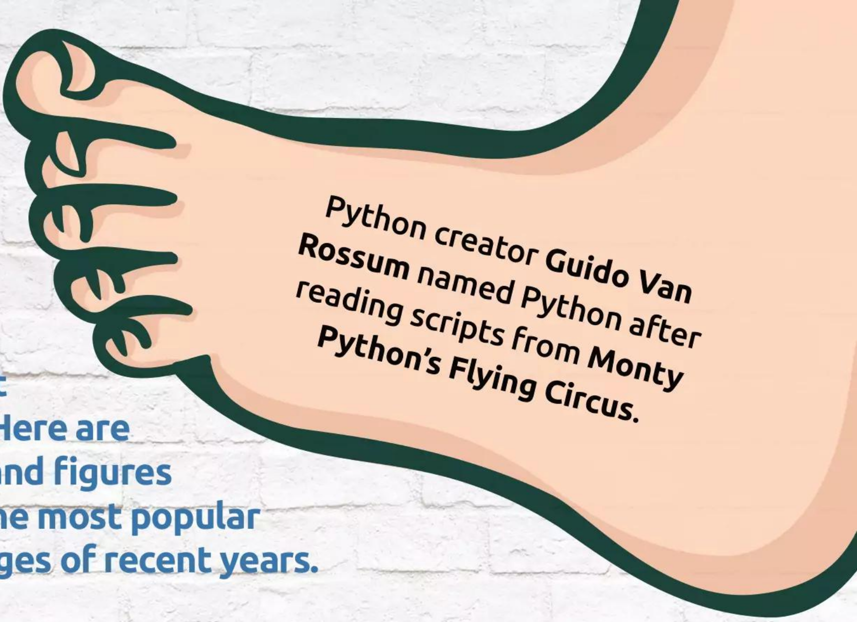


As you can see, Python is quite a versatile programming language. By learning Python, you are creating a well-rounded skillset that's able to take you into the next generation of computing, either professionally or simply as a hobbyist.

Whatever route you decide to take on your coding journey, you will do well to have Python in your corner.

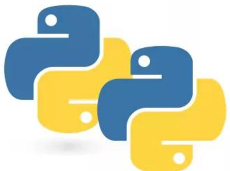
PYTHON in NUMBERS

There's a lot to like about Python, but don't just take our word for it. Here are some amazing facts and figures surrounding one of the most popular programming languages of recent years.



Python creator **Guido Van Rossum** named Python after reading scripts from **Monty Python's Flying Circus.**

Alexa, Amazon's Virtual Personal Assistant, uses Python to help with speech recognition.



Data analysis and Machine Learning are the two most used Python examples.

DECEMBER
2018

As of the end of 2018, Python was the most discussed language on the Internet.

PYTHON AND LINUX SKILLS ARE THE THIRD MOST POPULAR I.T. SKILLS IN THE UK.

Disney Pixar uses Python in its Renderman software to operate between other graphics packages.

OVER 75% OF RECOMMENDED CONTENT FROM NETFLIX IS GENERATED FROM MACHINE LEARNING – CODED BY PYTHON.

90% OF ALL FACEBOOK POSTS ARE FILTERED THROUGH PYTHON-CODED MACHINE LEARNING.

IT'S ESTIMATED THAT OVER 75% OF NASA'S WORKFLOW AUTOMATION SYSTEMS ON BOARD THE I.S.S. USE PYTHON.

16,000

There are over 16,000 Python jobs posted every six months in the UK.

PYTHON SKILL-BASED POSITIONS ARE THE 16th MOST SOUGHT-AFTER JOBS IN THE UK.

Python Data Science is thought to become the most sought-after job in the coming years.

Google is the top company for hiring Python developers, closely followed by Microsoft.

Data Science, Blockchain and Machine Learning are the fastest growing Python coding skills.

New York and San Francisco are the top Python developer cities in the world.

Python developers enjoy an average salary of

£60,000

95% OF ALL BEGINNER CODERS START WITH AND STILL USE, PYTHON AS THEIR PRIMARY OR SECONDARY LANGUAGE.

75% OF ALL PYTHON DEVELOPERS USE PYTHON 3, WHEREAS 25% STILL USE THE OUTDATED PYTHON 2 VERSION.

79% OF ALL PROGRAMMERS USE PYTHON REGULARLY, 21% USE IT AS A SECONDARY LANGUAGE.

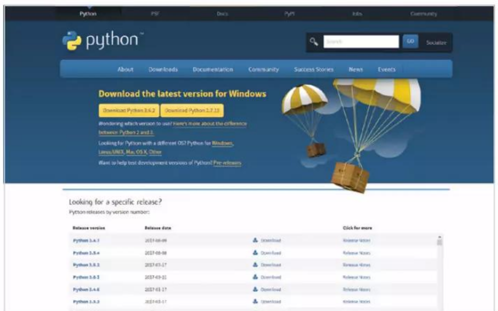
49% OF WINDOWS 10 DEVELOPERS USE PYTHON 3 AS THEIR MAIN PROGRAMMING LANGUAGE.

Equipment You Will Need

You can learn Python with very little hardware or initial financial investment. You don't need an incredibly powerful computer and any software that's required is freely available.

WHAT WE'RE USING

Thankfully, Python is a multi-platform programming language available for Windows, macOS, Linux, Raspberry Pi and more. If you have one of those systems, then you can easily start using Python.


Python
PSF
Docs
PyPI
Jobs
Community
python
GO
Socialize
About
Downloads
Documentation
Community
Success Stories
News
Events
Download Python 3.6.2
Download Python 2.7.13

Download the latest version for Windows

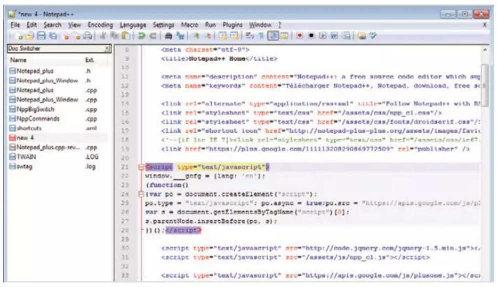
Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

Looking for a specific release?

Python releases by version number:


*new 4 - Notepad++
File
Edit
Search
View
Encoding
Language
Settings
Macro
Run
Plugins
Window
?
Doc Switcher
Name
Ext.
Notepad_plus .h
Notepad_plus_Window .h
Notepad_plus .cpp
Notepad_plus_Window .cpp
NppBigSwitch .cpp
NppCommands .cpp
shortcuts .xml
new 4
Notepad_plus.cpp-rev... .cpp
TWAIN .LOG
swtag .log

```
    <meta charset="utf-8">
    <title>Notepad++ Home</title>

    <meta name="description" content="Notepad++: a free source code editor which su
    <meta name="keywords" content="Télécharger Notepad++, Notepad, download, free s
    
    <link rel="alternate" type="application/rss+xml" title="Follow Notepad++ with R
    <link rel="stylesheet" type="text/css" href="/assets/css/npp_cl.css"/>
    <link rel="stylesheet" type="text/css" href="/assets/css/fonts/droidserif.css"/
    <link rel="shortcut icon" href="http://notepad-plus-plus.org/assets/images/favi
    <!--[if lte IE 7]><link rel="stylesheet" type="text/css" href="/assets/css/ie67
    <link href="https://plus.google.com/111113208290869772509" rel="publisher" />

<script type="text/javascript">
window.___gcfg = {lang: 'en'};
(function()
{var po = document.createElement("script");
po.type = "text/javascript"; po.async = true;po.src = "https://apis.google.com/js/p
var s = document.getElementsByTagName("script")[0];
s.parentNode.insertBefore(po, s);
})();</script>

    <script type="text/javascript" src="http://code.jquery.com/jquery-1.5.min.js"><
    <script type="text/javascript" src="/assets/js/npp_cl.js"></script>

    <script type="text/javascript" src="https://apis.google.com/js/plusone.js"></sc
```

☐ COMPUTER

Obviously you're going to need a computer in order to learn how to program in Python and to test your code. You can use Windows (from XP onward) on either a 32 or 64-bit processor, an Apple Mac or Linux installed PC.

☐ AN IDE

An IDE (Integrated Developer Environment) is used to enter and execute Python code. It enables you to inspect your program code and the values within the code, as well as offering advanced features. There are many different IDEs available, so find the one that works for you and gives the best results.

☐ PYTHON SOFTWARE

macOS and Linux already come with Python preinstalled as part of the operating system, as does the Raspberry Pi. However, you need to ensure that you're running the latest version of Python. Windows users need to download and install Python, which we'll cover shortly.

☐ TEXT EDITOR

Whilst a text editor is an ideal environment to enter code into, it's not an absolute necessity. You can enter and execute code directly from the IDLE but a text editor, such as Sublime Text or Notepad++, offers more advanced features and colour coding when entering code.

☐ INTERNET ACCESS

Python is an ever evolving environment and as such new versions often introduce new concepts or change existing commands and code structure to make it a more efficient language. Having access to the Internet will keep you up-to-date, help you out when you get stuck and give access to Python's immense number of modules.

☐ TIME AND PATIENCE

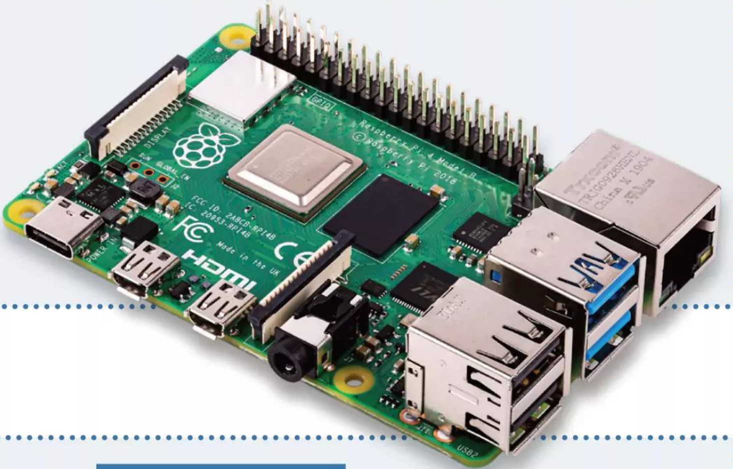
Despite what other books may lead you to believe, you won't become a programmer in 24-hours. Learning to code in Python takes time, and patience. You may become stuck at times and other times the code will flow like water. Understand you're learning something entirely new, and you will get there.

THE RASPBERRY PI

Why use a Raspberry Pi? The Raspberry Pi is a tiny computer that's very cheap to purchase, but offers the user a fantastic learning platform. Its main operating system, Raspbian, comes preinstalled with the latest Python along with many modules and extras.

RASPBERRY PI

The Raspberry Pi 4 Model B is the latest version, incorporating a more powerful CPU, a choice of 1GB, 2GB or 4GB memory versions and Wi-Fi and Bluetooth support. You can pick up a Pi from around £34, increasing up to £54 for the 4GB memory version, or as a part of kit for £50+, depending on the kit you're interested in.



FUZE PROJECT

The FUZE is a learning environment built on the latest model of the Raspberry Pi. You can purchase the workstations that come with an electronics kit and even a robot arm for you to build and program. You can find more information on the FUZE at **www.fuze.co.uk.**

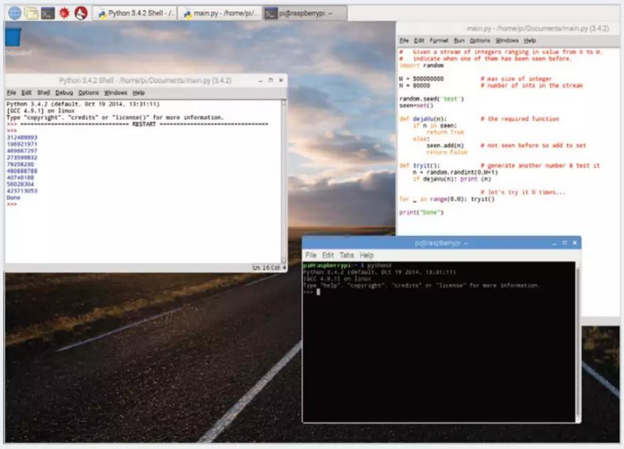
BOOKS

We have several great Raspberry Pi titles available via **www.bdmpublications.com**. Our Pi books cover how to buy your first Raspberry Pi, set it up and use it; there are some great step-by-step project examples and guides to get the most from the Raspberry Pi too.

NEW | Raspberry Pi 4 | Python | Code | Projects
BDM's Ultimate Series
Raspberry Pi
THE COMPLETE GUIDE
Unlock and master the power of the Raspberry Pi
NEW for 2020
Learn how to create amazing Pi projects
Boost your knowledge of coding
164 PAGES Raspberry Pi 4, Linux
BDM's Ultimate Series
Raspberry Pi
The Beginners' Guide
Everything you need to know about the Raspberry Pi
Over 874 Tutorials and Tips inside
Featuring Python, Linux, and Pi Projects!
Inside
Packed with top tips and tutorials
Learn Python 3 and Linux
Create amazing Pi projects
Unique type-in listings inside

RASPBIAN

The Raspberry Pi's main operating system is a Debian-based Linux distribution that comes with everything you need in a simple to use package. It's streamlined for the Pi and is an ideal platform for hardware and software projects, Python programming and even as a desktop computer.


Python 3.4.2 Shell - /...
main.py - /home/pi/...
pi@raspberrypi: ~
Python 3.4.2 Shell - /home/pi/Documents/main.py (3.4.2)
File Edit Shell Debug Options Windows Help
Ln: 16 Col: 4
main.py - /home/pi/Documents/main.py (3.4.2)
File Edit Format Run Options Windows Help
pi@raspberrypi: ~
File Edit Tabs Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
312489993
196921971
469667297
273599832
79258280
480888788
40748188
56028304
423713053
Done
>>>
```

```
#   Given a stream of integers ranging in value from 0 to M,
#   indicate when one of them has been seen before.
import random

M = 500000000           # max size of integer
N = 80000               # number of ints in the stream

random.seed('test')
seen=set()

def dejaVu(n):          # the required function
    if n in seen:
        return True
    else:
        seen.add(n)     # not seen before so add to set
        return False

def tryit():            # generate another number & test it
    n = random.randint(0,M+1)
    if dejaVu(n): print (n)

                        # let's try it N times...
for _ in range(0,N): tryit()

print("Done")
```

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Getting to Know Python

Python is one of the most used and popular programming languages. It's a great language on which to learn how to code, but it's also powerful enough to be used by companies when trawling through petabytes of raw data.

PYTHON POWER

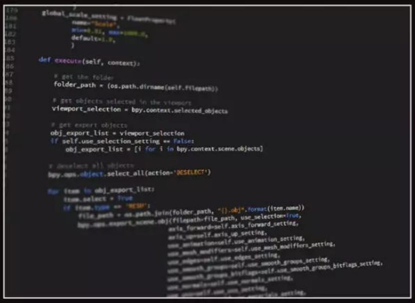
Python is a high-level, general-purpose programming language that was designed by Guido van Rossum in the late '80s, as a successor to the ABC Programming Language, and became available to use in 1990.

Guido van Rossum, the designer of Python.

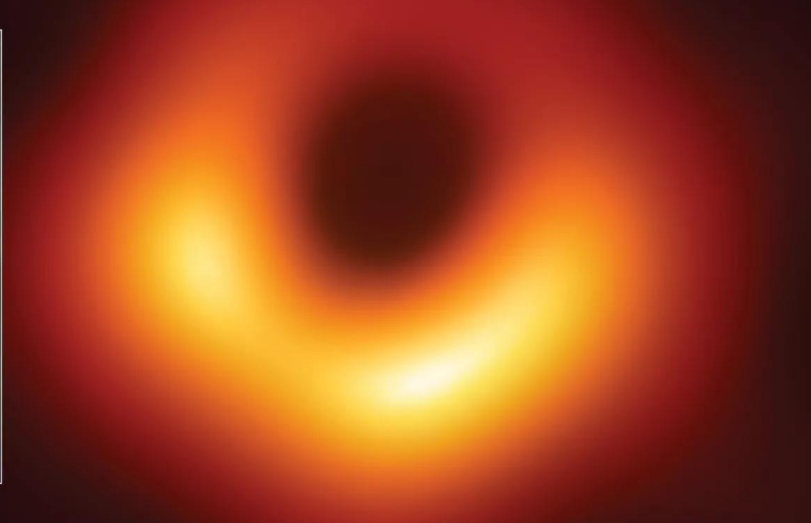
The Python Software Foundation, a non-profit organisation devoted to both furthering and improving Python, currently develops the language. The foundation's purpose is to "promote, protect, and advance the Python programming language, and to support and facilitate the growth of a diverse and international community of Python programmers."

The structure of Python code has been designed to flow easily, allowing those who are just beginning to code to follow it without too much difficulty. Yet, despite its ease of use, Python is regularly used throughout a number of industries in real-world scenarios.

Big Data and AI are the two fastest growing Python-backed technologies at the moment. Big Data is simply a modern term used to describe huge amounts of data, such as sequences of numbers collated during a day's worth of trading on the Stock Exchange. Python code is used to dig into the voluminous collection of numbers, and then feed back with reports on the highs and lows, what's trending, and so on.



Python code is both elegant and easy to read (when you know how).



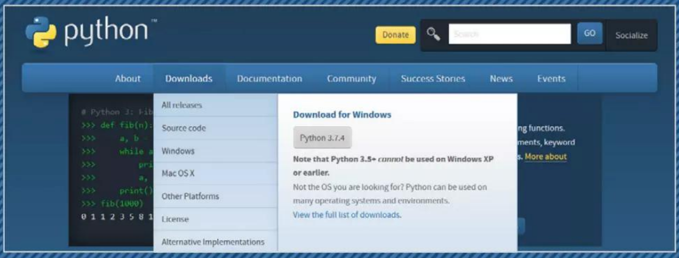
Made up from over 5 Petabytes of data, spread across a ton of hard drives, Python helped science to unveil the first image of a black hole.

As a side note, it's not just the likes of the Stock Exchange that use Python to study large quantities of data; in April 2019 the first image of a black hole was released, the supermassive black hole in the galaxy called M87, located roughly 55 million light years away. Thanks to the collaboration of over 200 scientists, using an array of telescopes spanning the world (called the Event Horizon Telescope Project), the combined power of the telescopes formed an impressive five petabytes of data, spread across tens of hard drives weighing in at nearly one ton. Five petabytes, by the way, equates to around 5,000 years' worth of MP3 files. Once all those hard drives were gathered together and shipped to a central supercomputer cluster, the team then used Python to painstakingly stitch together all the fragments of data from the five petabytes to finally form the most talked about astronomic event of the decade.

AI, if you're not familiar with the term, stands for Artificial Intelligence. Although we're still a long way off from the visionary stories of Arthur C. Clarke, AI is fast becoming one of the most influential technologies of our modern age. Rather than controlling robots, the AI that Python drives is designed to create autonomous ways of interacting with people online. For example, when you search for something on the Amazon website you will usually notice that similar products start to appear, whether within Amazon itself,

PYTHON 3 VS PYTHON 2

In a typical computing scenario, Python is complicated somewhat by the existence of two active versions of the language: Python 2 and Python 3.



Download for Windows

Note that Python 3.5+ *cannot* be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

Python 3 is the best option to download, or update to.

Python 3 is the newest release of the programming language. However, if you dig a little deeper into the Python site and investigate Python code online, you will undoubtedly come across Python 2. Crucially, although you can run Python 3 and Python 2 alongside each other, it's not recommended. Always opt for the latest stable release, as posted by the Python website. You will find, when using macOS or Linux, that Python 2 is already installed. This is because both these operating systems utilise elements of code necessary to the core functionality of the OS. Linux users tend to be better off, as most distributions package the latest version of Python 3 out-of-the-box, whereas macOS often has Python 3, it's usually an older version. Microsoft doesn't use any Python code for its core Windows systems, which is why you won't find Python inherent to Windows and therefore need to install it from scratch.

You need to be careful when you look up Python code online, although there are countless websites that offer quick tutorials, code snippets and support – and 99% of these site are a great help to those starting out with Python – a lot of the sites haven't been updated for some time, and as such use Python 2. If you enter Python 2 code into the Python 3 IDLE, the chances are it won't work due to incompatibilities between the older version and the newer. Python 2 is good, but Python 3 is better. You can obviously spend time converting the Python 2 code into version 3, but, to begin with, it's best to make sure that the code you're looking at is for the Python 3 libraries. Don't worry, though, all the code in this book is designed for Python 3, and that includes all the sub versions from Python 3.1 to the latest 3.x.

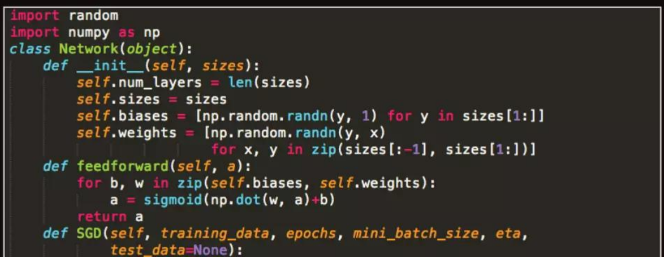
Python 3's growing popularity has meant that it's now prudent to start learning to develop with the new features and begin to phase out the previous version. Many development companies, such as SpaceX and NASA, use Python 3 for snippets of important code.

However, support for Python 2 is set to end on January 1st 2020, but this doesn't mean it'll be the last you see of it. Many Linux distros use Python 2 libraries, as does macOS, and to be fair, for the developers to transfer the existing Python 2 content to Python 3 may take more time than they have available, i.e. before the start of 2020. It's likely then, that we will still be seeing Python 2 long after it has had the final nail hammered into its coffin – in fact, expect to see that cut-off date extend further into the future.

or from a search engine, or even Facebook. The code behind these targeted snippets is Python, and it's using a form of AI to help determine what it is you would likely search for.

Despite the fact that many people find the targeting of advertising intrusive, or even an invasion of privacy, you have to admit that the code technology behind it all is rather impressive. With some very clever techniques, a Python developer is able to create a machine thinking script that not only displays items, news stories, books, other websites and ideas relating to what you've searched for, but it can also predict what you may be interested in looking for in the future. Another element to consider, with regards to AI, is that Python code can be used to help a computer learn how to do something more efficiently. In the case of neural networks in AI, the Python code is designed to return a result, then, as the code is run over and over again, the AI portion will begin to learn how to obtain a more accurate result, or do the maths behind the code quicker. It all depends on what the developer wants from their AI Python code.

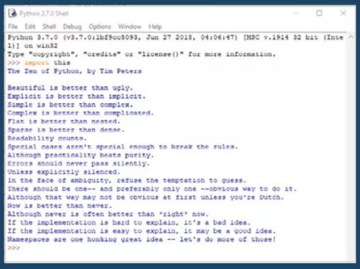
An example of Python AI code, using a feedforward neural network.



```
import random
import numpy as np
class Network(object):
    def __init__(self, sizes):
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x)
                        for x, y in zip(sizes[:-1], sizes[1:])]
    def feedforward(self, a):
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a)+b)
        return a
    def SGD(self, training_data, epochs, mini_batch_size, eta,
            test_data=None):
```

ZEN OF PYTHON

Python lets you access all the power of a computer in a language that humans can understand. Behind all this is an ethos called "The Zen of Python". This is a collection of 20 software principles that influences the design of the language. Principles include "Beautiful is better than ugly" and "Simple is better than complex." Type: `import this` into Python and it will display all the principles.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

The Zen of Python, as seen when entering: import this, into the Python IDLE.

As you will discover over the coming pages, Python is a fantastic language to learn. Get to grips with the basics, and before long, you'll be creating your own Python code for games, tools, and maybe even something in AI. The only limit with Python is your own imagination.

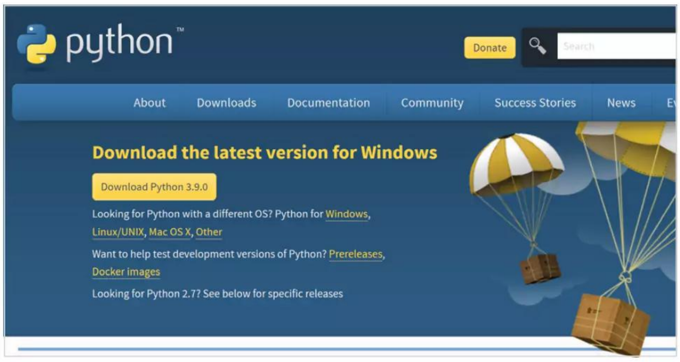
How to Set Up Python in Windows

Windows users can easily install the Python programming language via the main Python Downloads page. While most seasoned Python developers may shun Windows as the platform of choice for building their code, it's still an ideal starting point for beginners.

INSTALLING PYTHON 3.X

Microsoft Windows doesn't come with Python pre-installed as standard, so you're going to have to install it yourself manually. Thankfully, it's an easy process to follow.

STEP 1 Start by opening your web browser to **www.python.org/downloads/**. Look for the button detailing the Download link for Python 3.x. Python is regularly updated, changing the last digit for each bug fix and update. Therefore, don't worry if you see Python 3.7.0, or more, as long as it's Python 3, the code in this book will work fine.



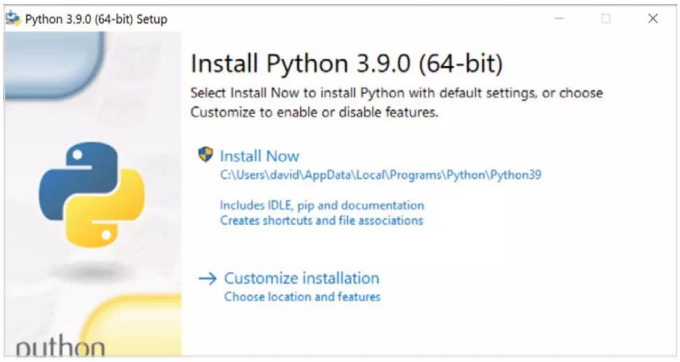
Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Prereleases, Docker images

Looking for Python 2.7? See below for specific releases

STEP 2 Click the download button for version 3.x, and save the file to your Downloads folder. When the file is downloaded, double-click the executable and the Python installation wizard will launch. From here you'll have two choices: Install Now and Customise Installation. We recommend opting for the Customise Installation link, and that the Add Python 3.x to PATH option is ticked.



Install Python 3.9.0 (64-bit)

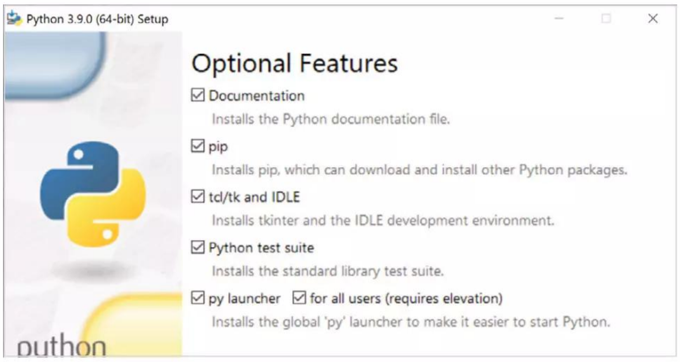
Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\david\AppData\Local\Programs\Python\Python39

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

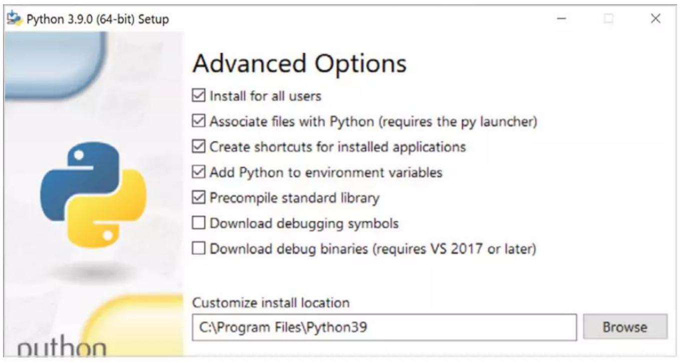
STEP 3 Choosing the Customise option allows you to specify certain parameters, and while you may stay with the defaults it's a good habit to adopt as sometimes (not with Python, thankfully) installers can include unwanted additional features. On the first screen available, ensure all boxes are ticked and click the Next button.



Optional Features

☑ Documentation
Installs the Python documentation file.
☑ pip
Installs pip, which can download and install other Python packages.
☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.
☑ Python test suite
Installs the standard library test suite.
☑ py launcher ☑ for all users (requires elevation)
Installs the global 'py' launcher to make it easier to start Python.

STEP 4 The next page of options include some interesting additions to Python. Ensure the Associate file with Python, Create Shortcuts, Add Python to Environment Variables, Precompile Standard Library, and Install for All Users options are ticked – these make using Python later much easier. Click Install when you're ready to continue.



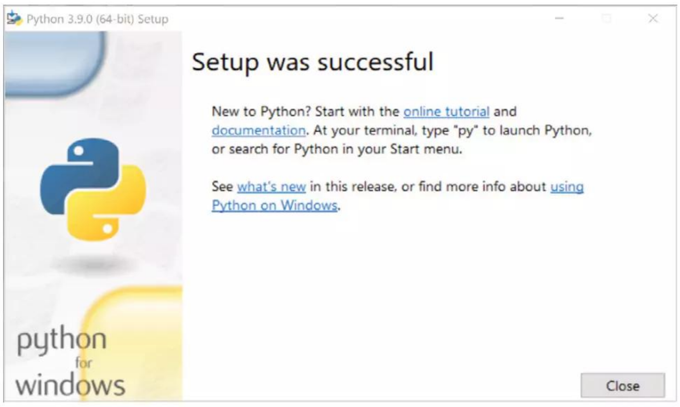
Advanced Options

☑ Install for all users
☑ Associate files with Python (requires the py launcher)
☑ Create shortcuts for installed applications
☑ Add Python to environment variables
☑ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2017 or later)

Customize install location
C:\Program Files\Python39

STEP 5

You may need to confirm the installation with the Windows authentication notification. Simply click Yes and Python will begin to install. Once the installation is complete the final Python wizard page will allow you to view the latest release notes, and follow some online tutorials.


Python 3.9.0 (64-bit) Setup
python for windows
Close

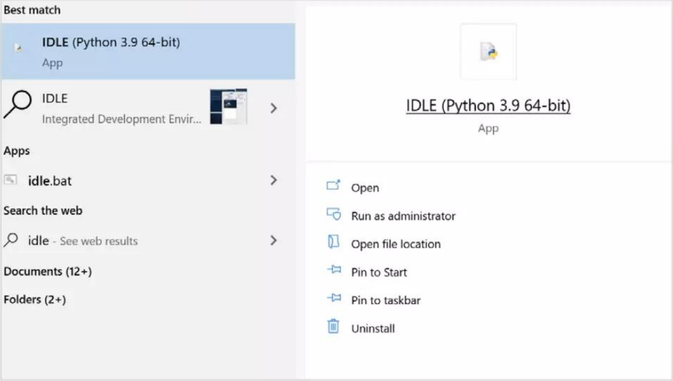
Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

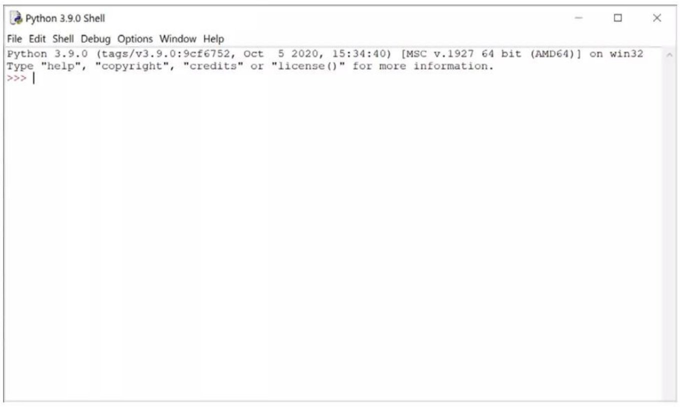
STEP 6

Windows 10 users will now find the installed Python 3.x within the Start button Recently Added section. The first link, Python 3.x (64-bit) – or 32-Bit (more on that in a moment) - will launch the command line version of Python when clicked (more on that in a moment). To open the IDLE, type IDLE into Windows start.


Best match
IDLE (Python 3.9 64-bit)
App
IDLE
Integrated Development Envir...
Apps
idle.bat
Search the web
idle - See web results
Documents (12+)
Folders (2+)
IDLE (Python 3.9 64-bit)
App
Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
Uninstall

STEP 7

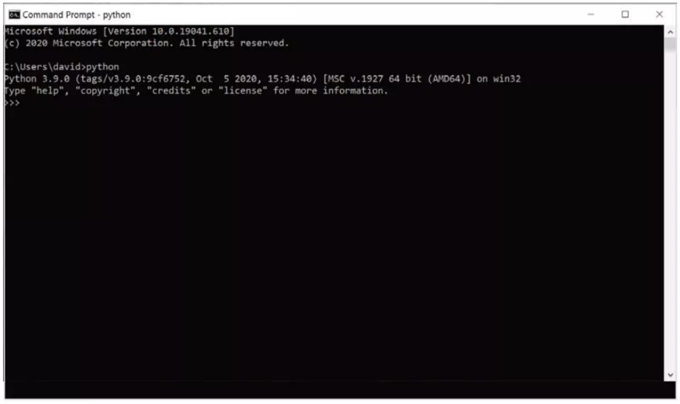
Clicking on the IDLE (Python 3.x 64-bit) link will launch the Python Shell, where we'll begin our Python programming journey. Don't worry if your version is newer, as long as it's Python 3.x our code will work inside your Python 3 interface.


Python 3.9.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

STEP 8

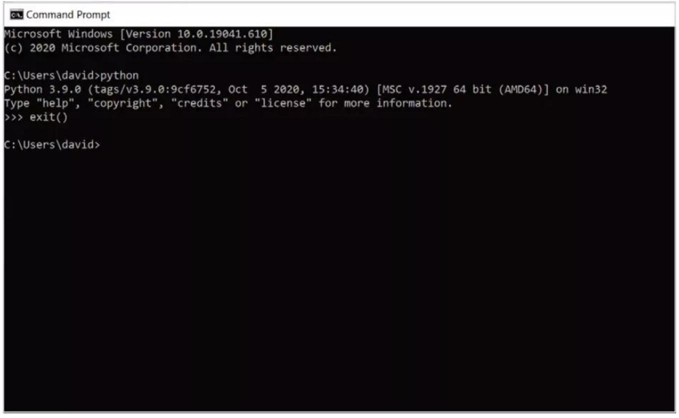
If you now click on the Windows Start button again, and this time type CMD you'll be presented with the Command Prompt link. Click it and you'll be at the Windows command line environment. To enter Python within the command line, you need to type: `python` and press Enter.


Command Prompt - python

```
Microsoft Windows [Version 10.0.19041.610]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\david>python
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

STEP 9

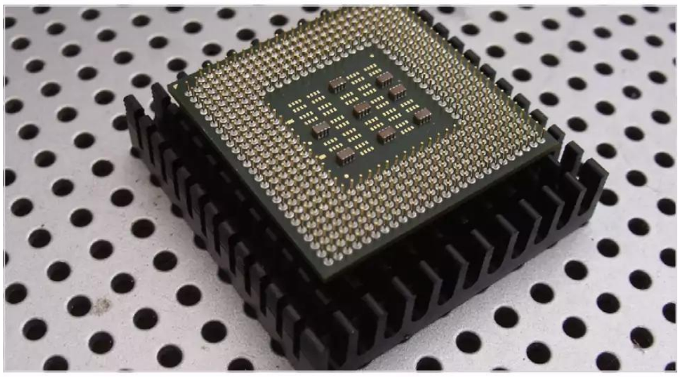
The command line version of Python works in much the same way as the Shell you opened in Step 7; note the three left-facing arrows (>>>). While it's a perfectly fine environment, it's not too user-friendly, so we'll leave the command line for now. Enter `exit()` to leave, and close the Command Prompt window.


Command Prompt

```
Microsoft Windows [Version 10.0.19041.610]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\david>python
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

C:\Users\david>
```

STEP 10

There are two versions of the latest Python available, 32-Bit and 64-Bit. Essentially, the 64-Bit version of Python allows you to utilise more than 4GB of memory. While this is great for users with a lot of memory, and memory hungry processes, it's not an essential feature of Python. However, the 32-bit version may provide more compatibility with existing modules.



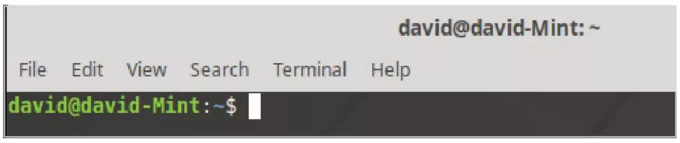
How to Set Up Python in Linux

While the Raspberry Pi's operating system contains the latest, stable version of Python, other Linux distros don't come with Python 3 pre-installed. If you're not going down the Pi route, then here's how to check and install Python for Linux.

PYTHON PENGUIN

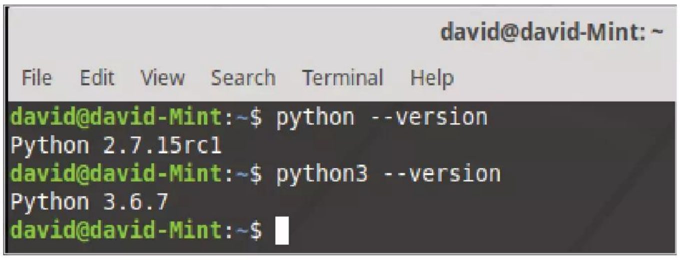
Linux is such a versatile operating system that it's often difficult to nail down just one-way of doing something. Different distributions go about installing software in different ways, so for this particular tutorial we will stick to Linux Mint.

STEP 1 First you need to ascertain which version of Python is currently installed in your Linux system. To begin with, drop into a Terminal session from your distro's menu, or hit the Ctrl+Alt+T keys.



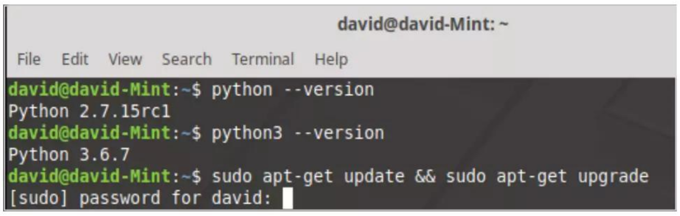
```
david@david-Mint:~$
```

STEP 2 Next, enter: `python --version` into the Terminal screen. You should have the output relating to version 2.x of Python in the display. Most Linux distro come with both Python 2 and 3 by default, as there's plenty of code out there still available for Python 2. Now enter: `python3 --version`.



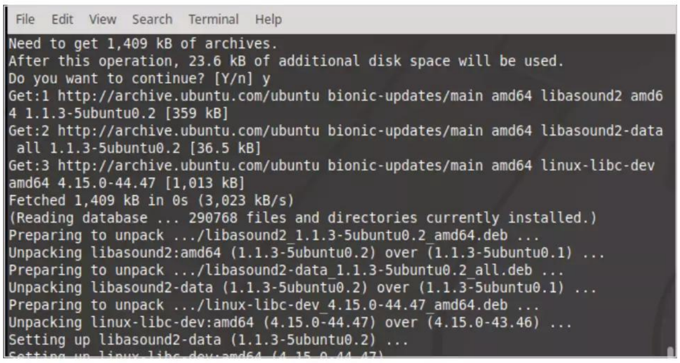
```
david@david-Mint:~$ python --version
Python 2.7.15rc1
david@david-Mint:~$ python3 --version
Python 3.6.7
david@david-Mint:~$
```

STEP 3 In our case we have both Python 2 and 3 installed. As long as Python 3.x.x is installed, then the code in our tutorials will work. It's always worth checking to see if the distro has been updated with the latest versions, enter: `sudo apt-get update && sudo apt-get upgrade` to update the system.



```
david@david-Mint:~$ python --version
Python 2.7.15rc1
david@david-Mint:~$ python3 --version
Python 3.6.7
david@david-Mint:~$ sudo apt-get update && sudo apt-get upgrade
[sudo] password for david:
```

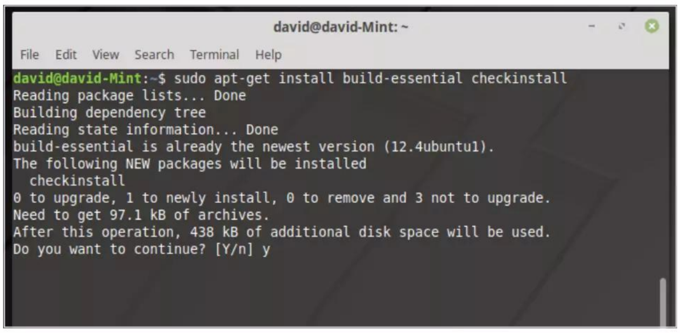
STEP 4 Once the update and upgrade completes, enter: `python3 --version` again to see if Python 3.x is updated, or even installed. As long as you have Python 3.x, you're running the most recent major version, the numbers after the 3. indicate patches plus further updates. Often they're unnecessary, but they can contain vital new elements.



```
Need to get 1,409 kB of archives.
After this operation, 23.6 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libasound2 amd6
4 1.1.3-5ubuntu0.2 [359 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libasound2-data
 all 1.1.3-5ubuntu0.2 [36.5 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 linux-libc-dev
amd64 4.15.0-44.47 [1,013 kB]
Fetched 1,409 kB in 0s (3,023 kB/s)
(Reading database ... 290768 files and directories currently installed.)
Preparing to unpack .../libasound2_1.1.3-5ubuntu0.2_amd64.deb ...
Unpacking libasound2:amd64 (1.1.3-5ubuntu0.2) over (1.1.3-5ubuntu0.1) ...
Preparing to unpack .../libasound2-data_1.1.3-5ubuntu0.2_all.deb ...
Unpacking libasound2-data (1.1.3-5ubuntu0.2) over (1.1.3-5ubuntu0.1) ...
Preparing to unpack .../linux-libc-dev_4.15.0-44.47_amd64.deb ...
Unpacking linux-libc-dev:amd64 (4.15.0-44.47) over (4.15.0-43.46) ...
Setting up libasound2-data (1.1.3-5ubuntu0.2) ...
```

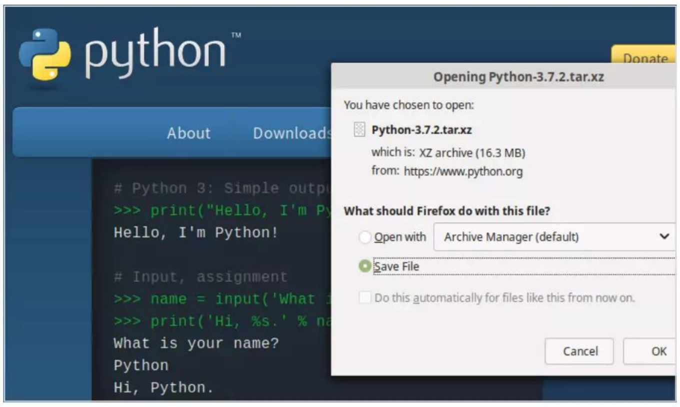
STEP 5 However, if you want the latest, cutting edge version, you'll need to build Python from source. Start by entering these commands into the Terminal:

```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev
libncursesw5-dev libssl-dev libsqlite3-dev tk-dev
libgdbm-dev libc6-dev libbz2-dev
```



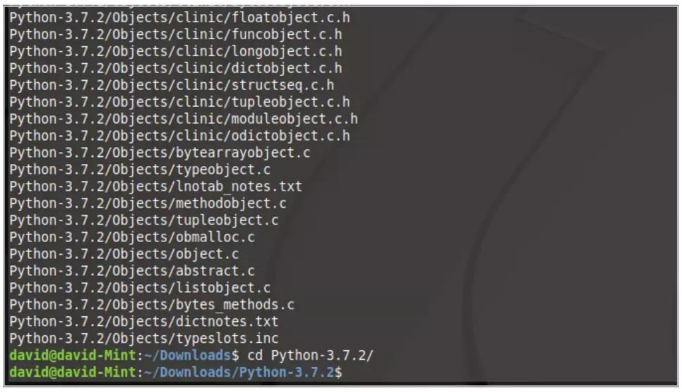
```
david@david-Mint:~$ sudo apt-get install build-essential checkinstall
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
The following NEW packages will be installed
  checkinstall
0 to upgrade, 1 to newly install, 0 to remove and 3 not to upgrade.
Need to get 97.1 kB of archives.
After this operation, 438 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

STEP 6 Open up your Linux web browser and go to the Python download page: **https://www.python.org/downloads**. Click on the Downloads, followed by the button under the Python Source window. This opens a download dialogue box, choose a location, then start the download process.


python™
Donate
About
Downloads
Opening Python-3.7.2.tar.xz
You have chosen to open:
Python-3.7.2.tar.xz
which is: XZ archive (16.3 MB)
from: https://www.python.org
What should Firefox do with this file?
○ Open with Archive Manager (default)
● Save File
☐ Do this automatically for files like this from now on.
Cancel
OK

```
# Python 3: Simple outp
>>> print("Hello, I'm Py
Hello, I'm Python!

# Input, assignment
>>> name = input('What i
>>> print('Hi, %s.' % na
What is your name?
Python
Hi, Python.
```

STEP 7 In the Terminal, go to the Downloads folder by entering: `cd Downloads/`. Then unzip the contents of the downloaded Python source code with: `tar -xvf Python-3.Y.Y.tar.xz` (replace the Y's with the version numbers you've downloaded). Now enter the newly unzipped folder with: `cd Python-3.Y.Y/`.

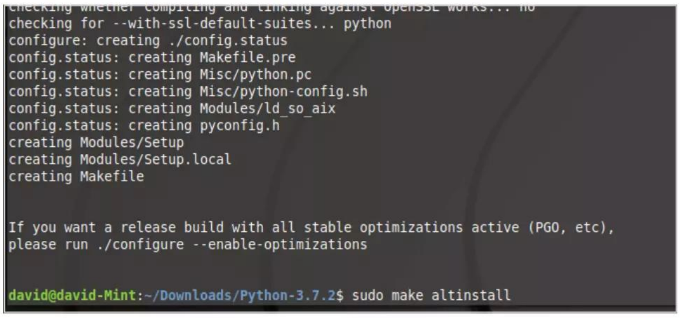


```
Python-3.7.2/Objects/clinic/floatobject.c.h
Python-3.7.2/Objects/clinic/funcobject.c.h
Python-3.7.2/Objects/clinic/longobject.c.h
Python-3.7.2/Objects/clinic/dictobject.c.h
Python-3.7.2/Objects/clinic/structseq.c.h
Python-3.7.2/Objects/clinic/tupleobject.c.h
Python-3.7.2/Objects/clinic/moduleobject.c.h
Python-3.7.2/Objects/clinic/odictobject.c.h
Python-3.7.2/Objects/bytearrayobject.c
Python-3.7.2/Objects/typeobject.c
Python-3.7.2/Objects/lnotab_notes.txt
Python-3.7.2/Objects/methodobject.c
Python-3.7.2/Objects/tupleobject.c
Python-3.7.2/Objects/obmalloc.c
Python-3.7.2/Objects/object.c
Python-3.7.2/Objects/abstract.c
Python-3.7.2/Objects/listobject.c
Python-3.7.2/Objects/bytes_methods.c
Python-3.7.2/Objects/dictnotes.txt
Python-3.7.2/Objects/typeslots.inc
david@david-Mint:~/Downloads$ cd Python-3.7.2/
david@david-Mint:~/Downloads/Python-3.7.2$
```

STEP 8 Within the Python folder, enter:

```
./configure
sudo make altinstall
```

This could take a while, depending on the speed of your computer. Once finished, enter: `python3.7 --version` to check the latest installed version. You now have Python 3.7 installed, alongside older Python 3.x.x and Python 2.

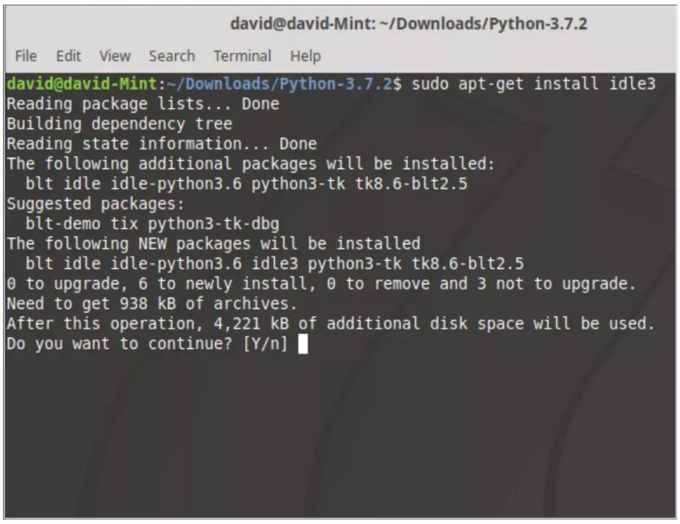


```
checking whether compiling and linking against OpenSSL works... no
checking for --with-ssl-default-suites... python
configure: creating ./config.status
config.status: creating Makefile.pre
config.status: creating Misc/python.pc
config.status: creating Misc/python-config.sh
config.status: creating Modules/ld_so_aix
config.status: creating pyconfig.h
creating Modules/Setup
creating Modules/Setup.local
creating Makefile


If you want a release build with all stable optimizations active (PGO, etc),
please run ./configure --enable-optimizations


david@david-Mint:~/Downloads/Python-3.7.2$ sudo make altinstall
```

STEP 9 For the GUI IDLE, you'll need to enter the following command into the Terminal:

```
sudo apt-get install idle3
```

The IDLE can then be started with the command: `idle3`. Note, that IDLE runs a different version to the one you installed from source.


david@david-Mint: ~/Downloads/Python-3.7.2
File Edit View Search Terminal Help

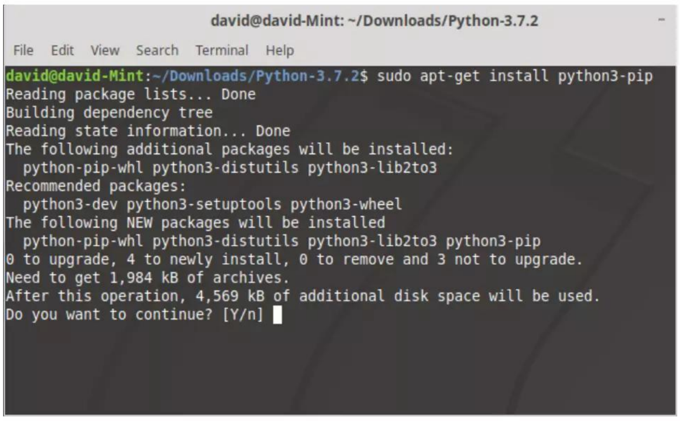
```
david@david-Mint:~/Downloads/Python-3.7.2$ sudo apt-get install idle3
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  blt idle idle-python3.6 python3-tk tk8.6-blt2.5
Suggested packages:
  blt-demo tix python3-tk-dbg
The following NEW packages will be installed
  blt idle idle-python3.6 idle3 python3-tk tk8.6-blt2.5
0 to upgrade, 6 to newly install, 0 to remove and 3 not to upgrade.
Need to get 938 kB of archives.
After this operation, 4,221 kB of additional disk space will be used.
Do you want to continue? [Y/n] █
```

STEP 10 You'll also need PIP (Pip Installs Packages), which is a tool to help you install more modules and extras. Enter: `sudo apt-get install python3-pip`

Once PIP is installed, check for the latest update with:

```
pip3 install --upgrade pip
```

When complete, close the Terminal and Python 3.x will be available via the Programming section in your distro's menu.


david@david-Mint: ~/Downloads/Python-3.7.2
File Edit View Search Terminal Help

```
david@david-Mint:~/Downloads/Python-3.7.2$ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  python-pip-whl python3-distutils python3-lib2to3
Recommended packages:
  python3-dev python3-setuptools python3-wheel
The following NEW packages will be installed
  python-pip-whl python3-distutils python3-lib2to3 python3-pip
0 to upgrade, 4 to newly install, 0 to remove and 3 not to upgrade.
Need to get 1,984 kB of archives.
After this operation, 4,569 kB of additional disk space will be used.
Do you want to continue? [Y/n] █
```

PYTHON ON macOS

Installation of Python on macOS can be done in much the same way as the Windows installation. Simply go to the Python webpage, hover your mouse pointer over the Downloads link and select Mac OS X from the options. You will then be guided to the Python releases for Mac versions, along with the necessary installers for macOS 64-bit for OS X 10.9 and later.

Python on the Pi

If you're considering on which platform to install and use Python, then give some thought to one of the best coding bases available: the Raspberry Pi. The Pi has many advantages for the coder: it's cheap, easy to use, and extraordinarily flexible.

THE POWER OF PI

While having a far more powerful coding platform on which to write and test your code is ideal, it's not often feasible. Most of us are unable to jump into a several hundred-pound investment when we're starting off and this is where the Raspberry Pi can help out.

While having a far more powerful coding platform on which to write and test your code is ideal, it's not often feasible. Most of us are unable to jump into a several hundred-pound investment when we're starting off and this is where the Raspberry Pi can help out.

The Raspberry Pi is a fantastic piece of modern hardware that has created, or rather re-created, the fascination we once all had about computers, how they work, how to code and foundation level electronics. Thanks to its unique mix of hardware and custom software, it has proved itself to be an amazing platform on which to learn how to code; specifically, using Python.

While you're able, with ease, to use the Raspberry Pi to learn to code with other programming languages, it's Python that has been firmly pushed to the forefront. The Raspberry Pi uses Raspbian as its recommended, default operating system. Raspbian is a Linux OS, or to be more accurate, it's a Debian-based distribution of Linux. This means that there's already a built-in element of Python programming, as opposed to a fresh installation of Windows 10, which has no Python-specific base. However, the Raspberry Pi Foundation has gone the extra mile to include a vast range of Python modules, extensions and even examples, out of the box. So, essentially, all you need to do is buy a Raspberry Pi, follow the instructions on how to set one up (by using one of our excellent Raspberry Pi guides found at **www.bdmpublications.com**) and you can start coding with Python as soon as the desktop has loaded.

Significantly, there's a lot more to the Raspberry Pi, which makes it an excellent choice for someone who is starting to learn how to code in Python. The Pi is remarkably easy to set up as a headless node. This means that, with a few tweaks here and there, you're able to remotely connect to the Raspberry Pi from any other computer, or device, on your home network. For example, once you've set up the remote connectivity options, you can simply plug the Pi into the power socket anywhere in your house within range of your wireless router. As long as the Pi is connected, you will be able to remotely access the desktop from Windows or macOS as easily as if you were sitting in front of the Pi with a keyboard and mouse.

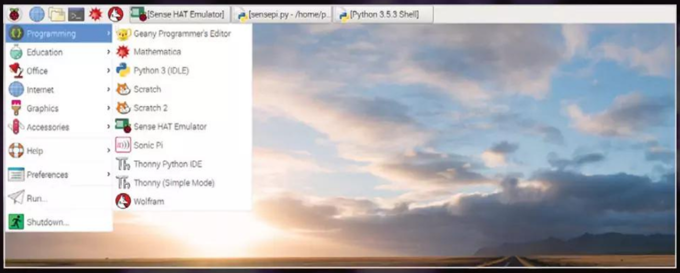
Using this method saves a lot of money, as you don't need another keyboard, mouse and monitor, plus, you won't need to allocate sufficient space to accommodate those extras either. If you're pushed for space and money, then for around £60, buying one of the many kits available will provide the Pi with a pre-loaded SD card (with the latest Raspbian OS), a case, power socket and cables, this is a good idea as you could, with very little effort, leave the Pi plugged into the wall under a desk, while still being able to connect to it and code.

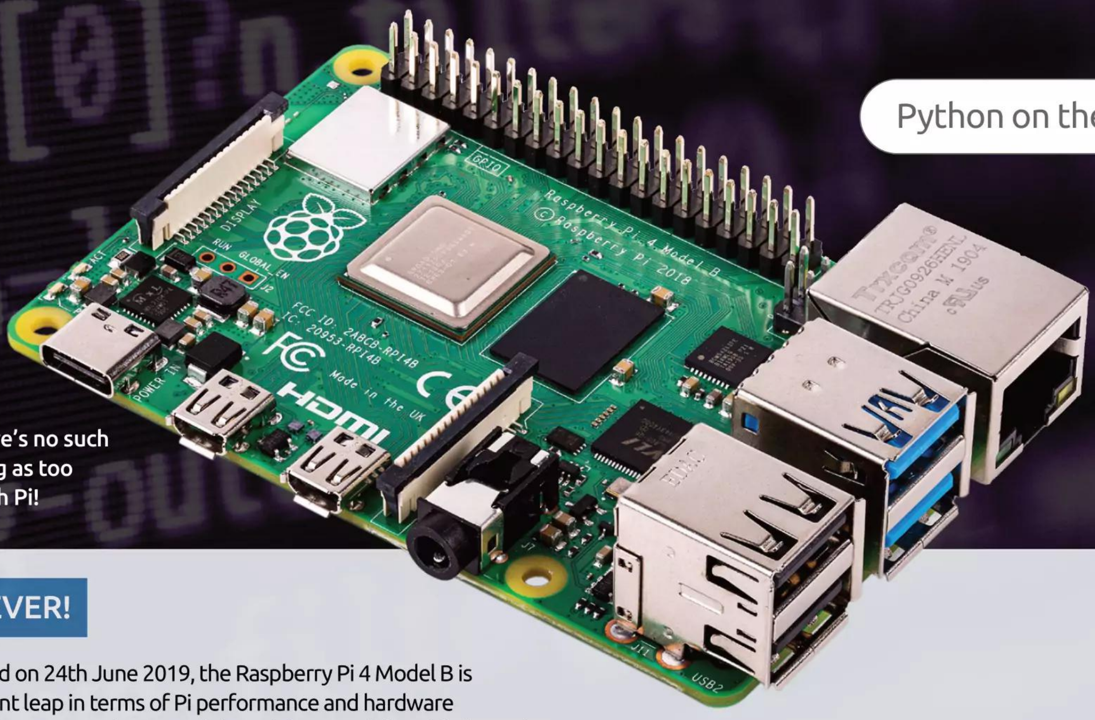
The main advantage is, of course, the extra content that the Raspberry Pi Foundation has included out of the box. The reason for this is that the Raspberry Pi's goal is to help educate the user, whether that's coding, electronics, or some other aspect of computing. To achieve that goal the Pi Foundation includes different IDEs for the user to compile Python code on; as well as both Python 2 and Python 3, there's even a Python library that allows you to communicate with Minecraft.

There are other advantages, such as being able to combine Python code with Scratch (an Object-Oriented programming language developed by MIT, for children to understand how coding works) and being able to code the GPIO connection on the Pi to further control any attached robotics or electronics projects. Raspbian also includes a Sense HAT Emulator (a HAT is a hardware attached piece of circuitry that offers different electronics, robotics and motorisation projects to the Pi), which can be accessed via Python code.

Consequently, the Raspberry Pi is an excellent coding base, as well as a superb project foundation. It is for these, and many other, reasons we've used the Raspberry Pi as our main Python codebase throughout this title. While the code is written and performed on a Pi, you're also able to use it in Windows, other versions of Linux and macOS. If the code requires a specific operating system, then, don't worry; we will let you know in the text.

Everything you need to learn to code with Python is included with the OS!




Raspberry Pi 4 Model B
© Raspberry Pi 2018
HDMI
FCC ID: 2ABCB-RPI4B
IC: 20953-RPI4B
Made in the UK
GPIO
DISPLAY
POWER IN
USB2

There's no such thing as too much Pi!

PI 4-EVER!

Introduced on 24th June 2019, the Raspberry Pi 4 Model B is a significant leap in terms of Pi performance and hardware specifications. It was also one of the quickest models, aside from the original Pi, to sell out.

With a new 1.5GHz, 64-bit, quad-core ARM Cortex-A72 processor, and a choice of 2GB, 4GB, or 8GB memory versions, the Pi 4 is onestep closer to becoming a true desktop computer. In addition, the Pi 4 was launched with the startling decision to include dual-monitor support, in the form of a pair of two micro-HDMI ports.

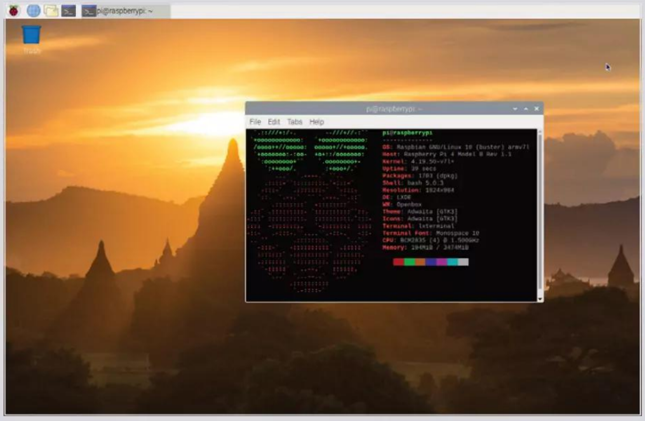
You'll also find a pair of USB 3.0 ports, Bluetooth 5.0, and a GPU that's capable of handing 4K resolutions and OpenGL ES 3.0 graphics. In short, the Pi 4 is the most powerful of the current Raspberry Pi models. However, the different memory versions have an increased cost. The 2GB version costs £34, 4GB is £54, and the 8GB Pi 4 will set you back £74. Remember to also factor in one or two micro-HDMI cables with your order.

RASPBIAN BUSTER

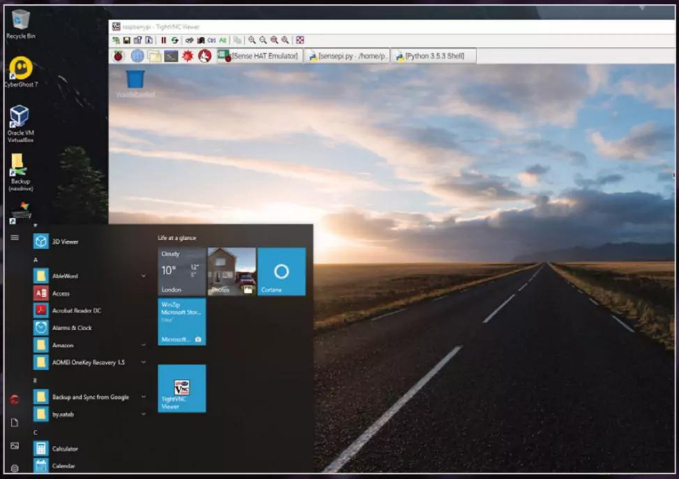
In addition to releasing the Pi 4, the Raspberry Pi team also compiled a new version of the Raspbian operating system, codenamed Buster.

In conjunction with the new hardware the Pi 4 boasts, Buster does offer a few updates. Although on the whole it's very similar in appearance and operation to the previous version of Raspbian. The updates are mainly in-line with the 4K's display and playback, giving the Pi 4 a new set of graphical drivers and performance enhancements.

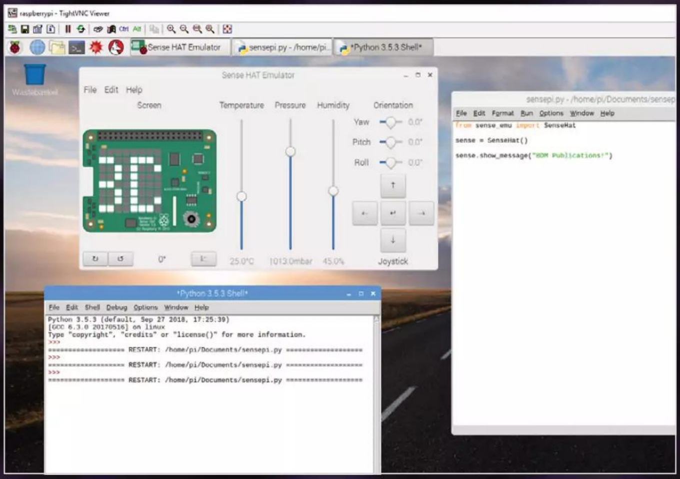
In short, what you see in this book will work with the Raspberry Pi 4 and Raspbian Buster!


pi@raspberrypi: ~
File Edit Tabs Help

Once set up, you can remotely connect to the Pi's desktop from any device/PC.


Recycle Bin
raspberrypi - TightVNC Viewer
Sense HAT Emulator
sensepi.py - /home/p...
Python 3.5.3 Shell
3D Viewer
Life at a glance
London
Cortana
Backup and Sync from Google
Calculator
Calendar
TightVNC Viewer

You can even test connected hardware with Python remotely, via Windows.


raspberrypi - TightVNC Viewer
Sense HAT Emulator
sensepi.py - /home/pi...
Python 3.5.3 Shell
File Edit Help
Screen
Temperature
Pressure
Humidity
Orientation
Yaw
Pitch
Roll
0.0°
0°
25.0°C
1013.0mbar
45.0%
Joystick
sensepi.py - /home/pi/Documents/sensep
File Edit Format Run Options Window Help
File Edit Shell Debug Options Window Help

```
from sense_emu import SenseHat

sense = SenseHat()

sense.show_message("BDM Publications!")
```

```
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: /home/pi/Documents/sensepi.py ====================
>>>
==================== RESTART: /home/pi/Documents/sensepi.py ====================
>>>
==================== RESTART: /home/pi/Documents/sensepi.py ====================
```

It’s a long way from creating your fist Python code to controlling a robot on Mars, but it’s not impossible. Every advanced and professional Python coder started with the base fundamentals, tips and fixes mentioned in this section.

With these foundation Python concepts, you will be able to advance your skills and branch out into whatever Python project you have in mind. From printing Hello World on the screen, looking into conditions and loops, asking a user for input and reacting to it, to using variables to control your code, this is where your Python skills are forged.

72 Starting Python for the First Time

74 Your First Code

76 Saving and Executing Your Code

78 Executing Code from the Command Line

80 Numbers and Expressions

82 Using Comments

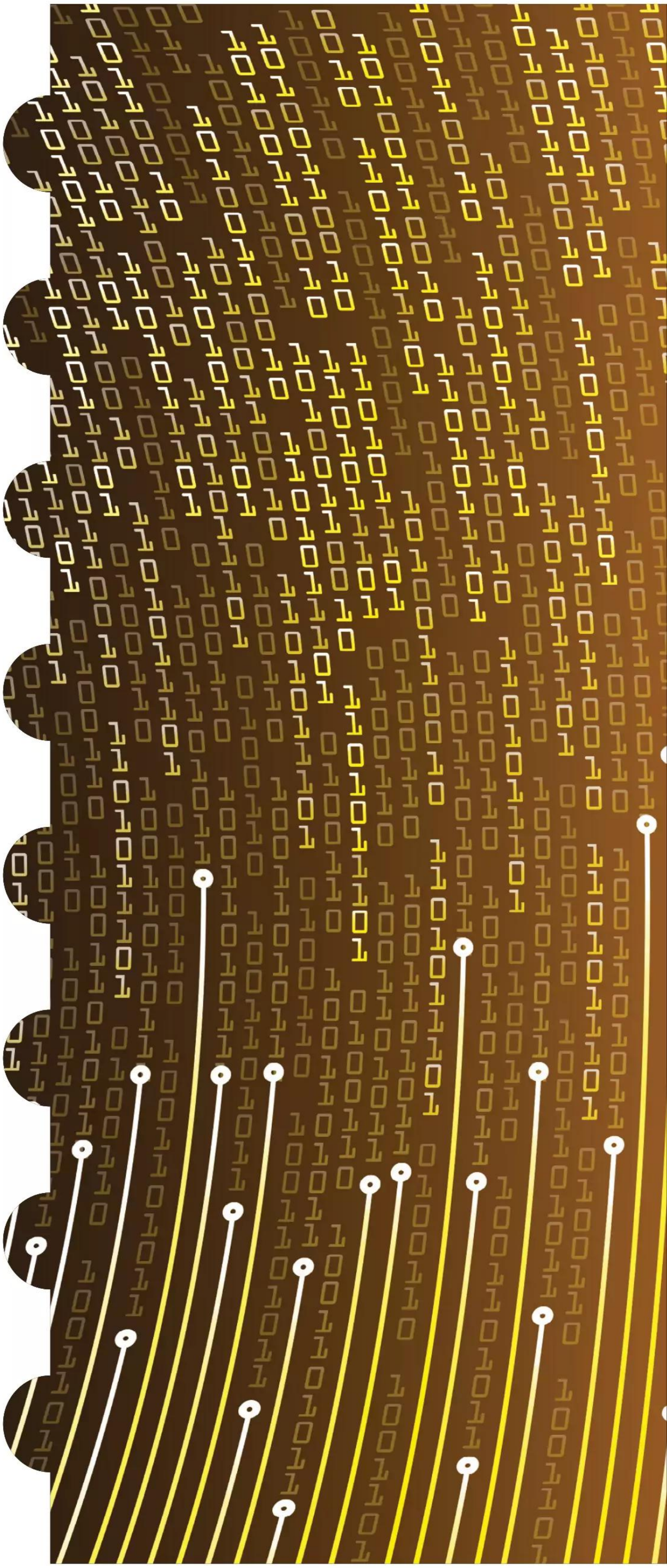
84 Working with Variables

86 User Input

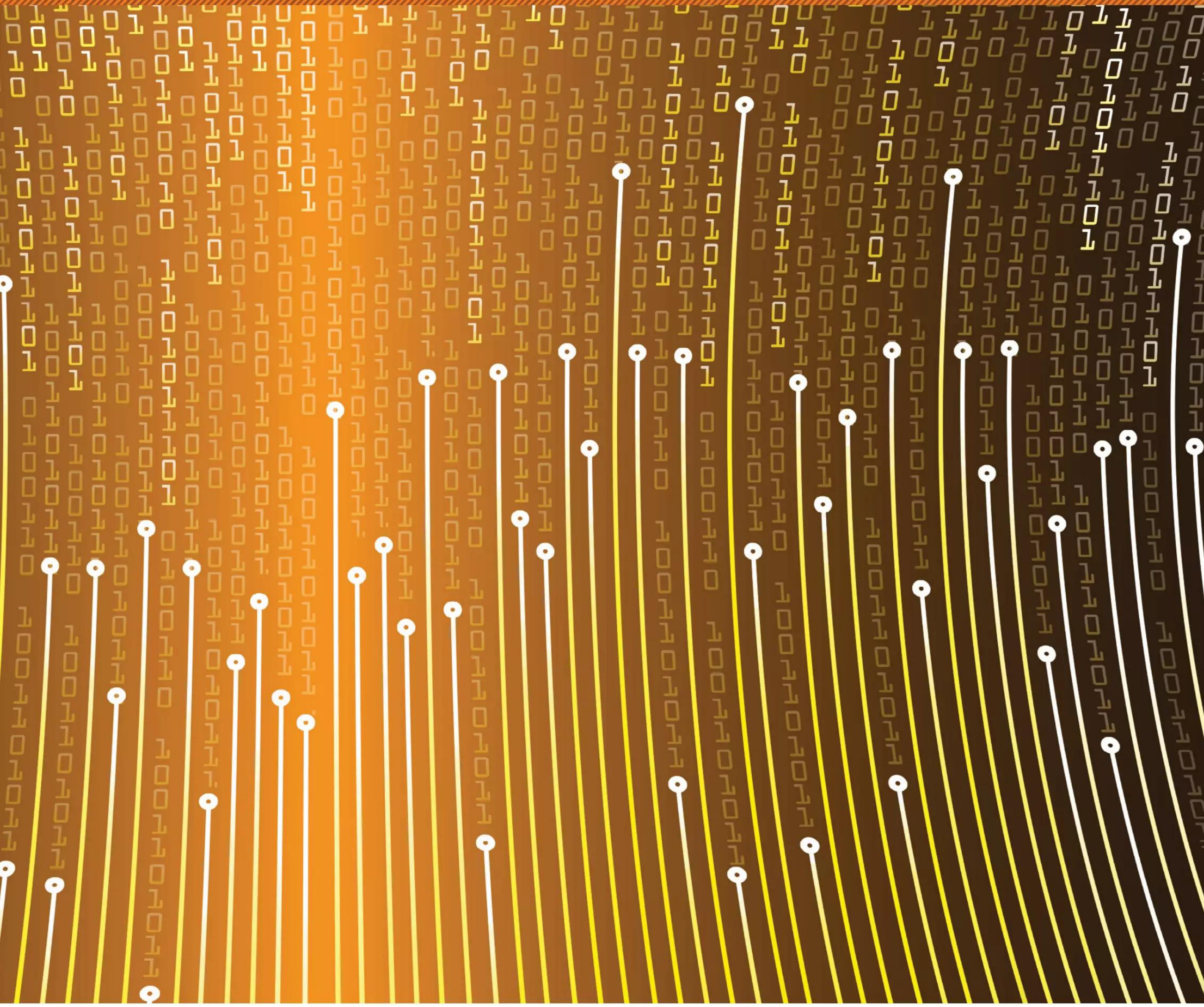
88 Creating Functions

90 Conditions and Loops

92 Python Modules



Coding with Python



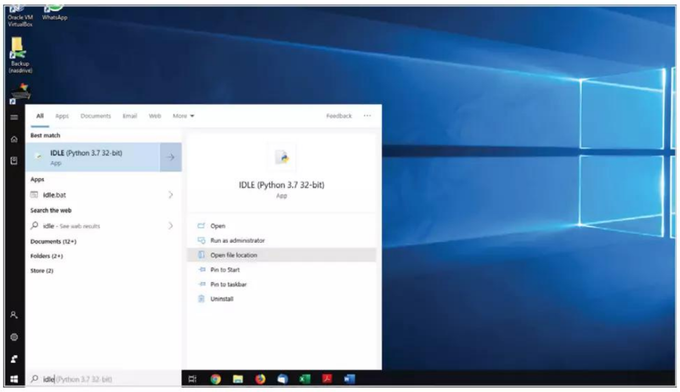
Starting Python for the First Time

We're using Python 3 under Windows 10 for these following examples. Don't worry if your version of Python is 3.4.2, or something lesser than the current version, as long as you're using Python 3, the code will work.

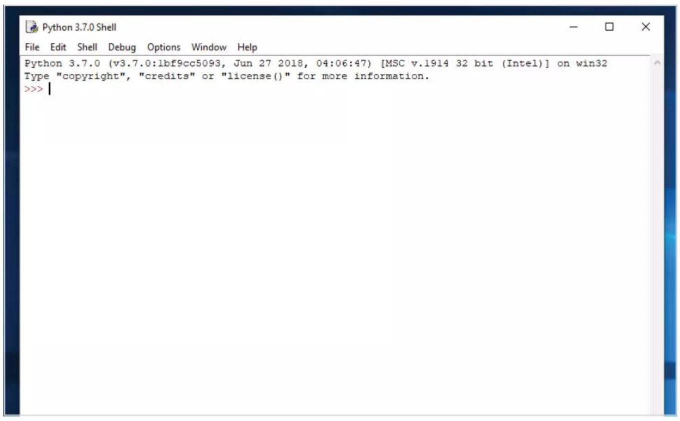
STARTING PYTHON

As when learning anything new, you need to start slow. You can pick up the pace as your experience grows, but for now, let's just get something appearing on the screen. Don't worry, you'll soon be coding like a pro!

STEP 1 Click on the Windows Start button, and start typing 'idle'. The result will be the currently installed version of the Python IDLE on your system. You can Pin it to the Start for convenience, otherwise simply click the icon to launch the Python Shell.

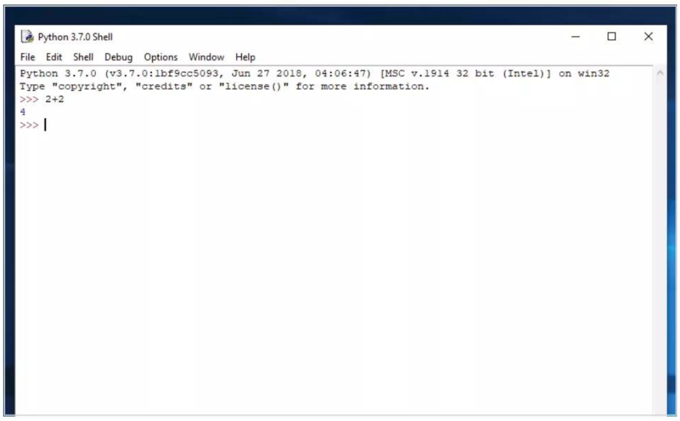

Oracle VM VirtualBox
WhatsApp
Backup (nasdrive)
All
Apps
Documents
Email
Web
More
Feedback
Best match
IDLE (Python 3.7 32-bit)
App
Apps
idle.bat
Search the web
idle - See web results
Documents (12+)
Folders (2+)
Store (2)
IDLE (Python 3.7 32-bit)
App
Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
Uninstall
idle

STEP 2 The Shell is where you can enter code and see the responses and output of code you've programmed into Python. This is a kind of sandbox, if you will, where you're able to try out some simple code and processes.


Python 3.7.0 Shell
File
Edit
Shell
Debug
Options
Window
Help

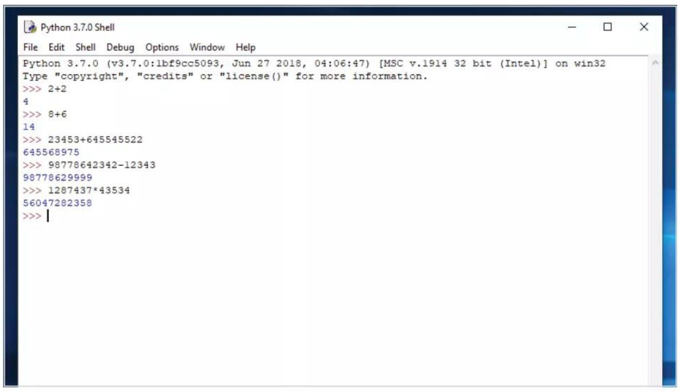
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
```

STEP 3 For example, in the Shell enter: `2+2` After pressing **Enter**, the next line will display the answer: 4. Basically, Python has taken the 'code' and produced the relevant output.


Python 3.7.0 Shell
File
Edit
Shell
Debug
Options
Window
Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 
```

STEP 4 The Python Shell acts very much like a calculator, since code is basically a series of mathematical interactions with the system. Integers, which are the infinite sequence of whole numbers, can easily be added, subtracted, multiplied, and so on.


Python 3.7.0 Shell
File
Edit
Shell
Debug
Options
Window
Help

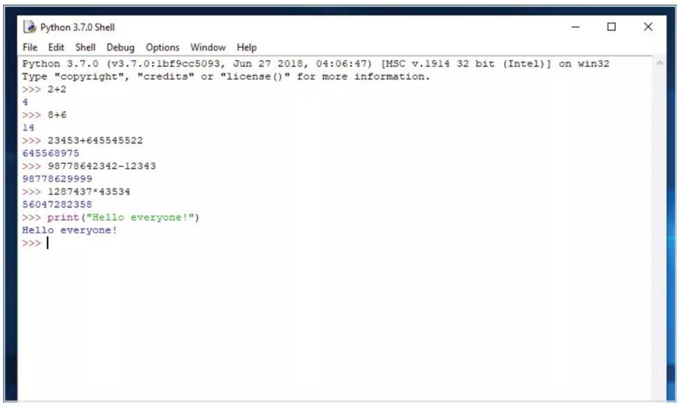
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+645545522
645568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> 
```

STEP 5

While that's very interesting, it's not particularly exciting. Instead, try this:

```
print("Hello everyone!")
```

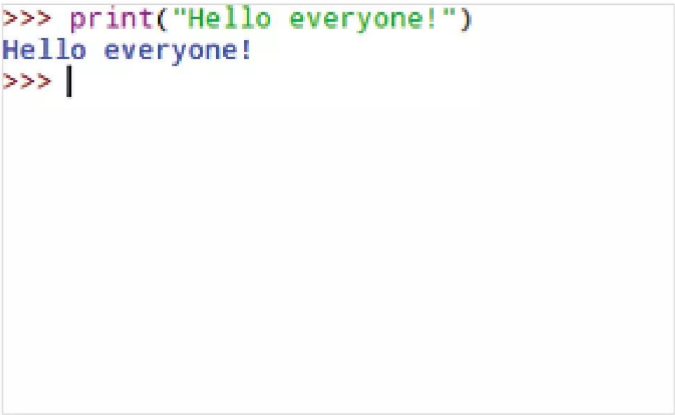
Just enter it into the IDLE as you've done in the previous steps.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+645545522
645568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> print("Hello everyone!")
Hello everyone!
>>> 
```

STEP 6

This is a little more like it, since you've just produced your first bit of code. The Print command is fairly self-explanatory, it prints things. Python 3 requires the parentheses as well as quotes in order to output content to the screen, in this case the 'Hello everyone!' bit.

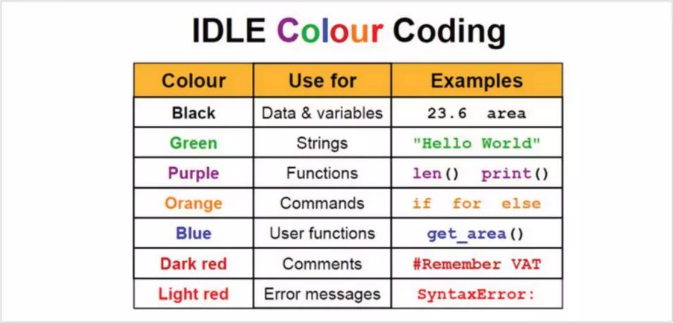


```
>>> print("Hello everyone!")
Hello everyone!
>>> 
```

STEP 7

You'll have noticed the colour coding within the Python IDLE. The colours represent different elements of Python code. They are:

Black – Data and Variables
Green – Strings
Purple – Functions
Orange – Commands
Blue – User Functions
Dark Red – Comments
Light Red – Error Messages

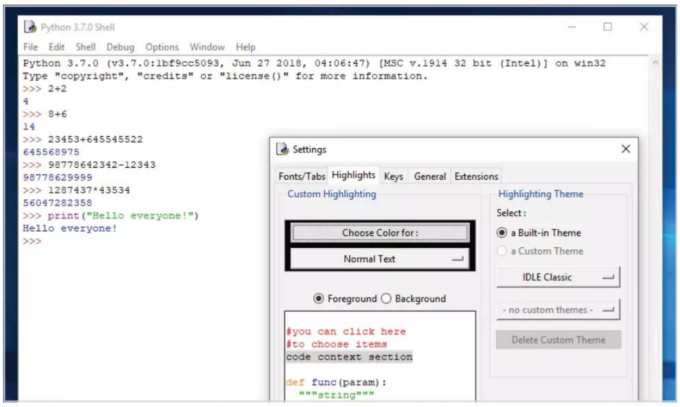


IDLE Colour Coding

Colour	Use for	Examples
Black	Data & variables	`23.6 area`
Green	Strings	`"Hello World"`
Purple	Functions	`len() print()`
Orange	Commands	`if for else`
Blue	User functions	`get_area()`
Dark red	Comments	`#Remember VAT`
Light red	Error messages	`SyntaxError:`

STEP 8

The Python IDLE is a configurable environment. If you don't like the way the colours are represented, then you can always change them via **Options > Configure IDLE**, and clicking on the **Highlighting** tab. However, we don't recommend that as you won't be seeing the same as our screenshots.



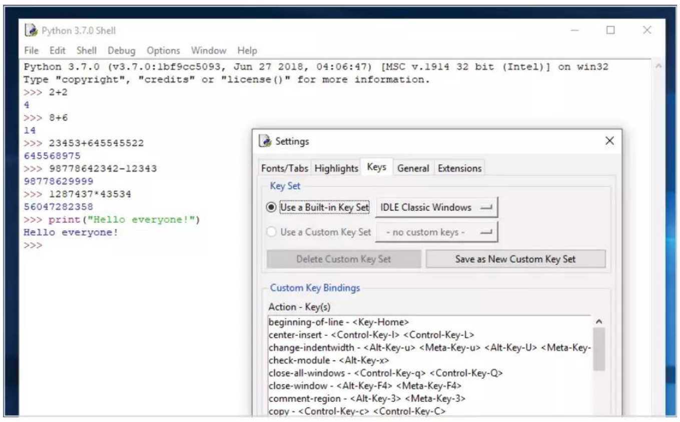
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+645545522
645568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> print("Hello everyone!")
Hello everyone!
>>>
```

Normal Text
● Foreground ○ Background
Select :
● a Built-in Theme
○ a Custom Theme
IDLE Classic
- no custom themes -

```
#you can click here
#to choose items
code context section

def func(param):
  """string"""
```

STEP 9

As with most programs available, regardless of the operating system, there are numerous shortcut keys. We don't have room for them all here, but within the Options > Configure IDLE and under the **Keys** tab, you'll see a list of the current bindings.



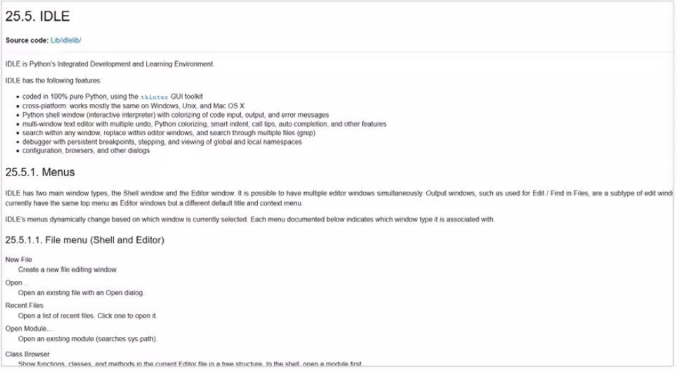
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+645545522
645568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> print("Hello everyone!")
Hello everyone!
>>>
```

● Use a Built-in Key Set: IDLE Classic Windows
○ Use a Custom Key Set: - no custom keys -

Action - Key(s)
beginning-of-line - <Key-Home>
center-insert - <Control-Key-l> <Control-Key-L>
change-indentwidth - <Alt-Key-u> <Meta-Key-u> <Alt-Key-U> <Meta-Key-
check-module - <Alt-Key-x>
close-all-windows - <Control-Key-q> <Control-Key-Q>
close-window - <Alt-Key-F4> <Meta-Key-F4>
comment-region - <Alt-Key-3> <Meta-Key-3>
copy - <Control-Key-c> <Control-Key-C>

STEP 10

The Python IDLE is a power interface, and one that's actually been written in Python using one of the available GUI toolkits. If you want to know the many ins and outs for the Shell, we recommend you take a few moments to view **https://docs.python.org/3/library/idle.html**, which details many of the IDLE's features.



25.5. IDLE

Source code: Lib/idlelib/

IDLE is Python's Integrated Development and Learning Environment.

IDLE has the following features:

- coded in 100% pure Python, using the `tkinter` GUI toolkit
- cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
- multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
- search within any window, replace within editor windows, and search through multiple files (grep)
- debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
- configuration, browsers, and other dialogs

25.5.1. Menus

IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously. Output windows, such as used for Edit / Find in Files, are a subtype of edit wind
currently have the same top menu as Editor windows but a different default title and context menu.

IDLE's menus dynamically change based on which window is currently selected. Each menu documented below indicates which window type it is associated with.

25.5.1.1. File menu (Shell and Editor)

New File
Create a new file editing window.
Open...
Open an existing file with an Open dialog.
Recent Files
Open a list of recent files. Click one to open it.
Open Module...
Open an existing module (searches sys.path).
Class Browser
Show functions, classes, and methods in the current Editor file in a tree structure. In the shell, open a module first.

Your First Code

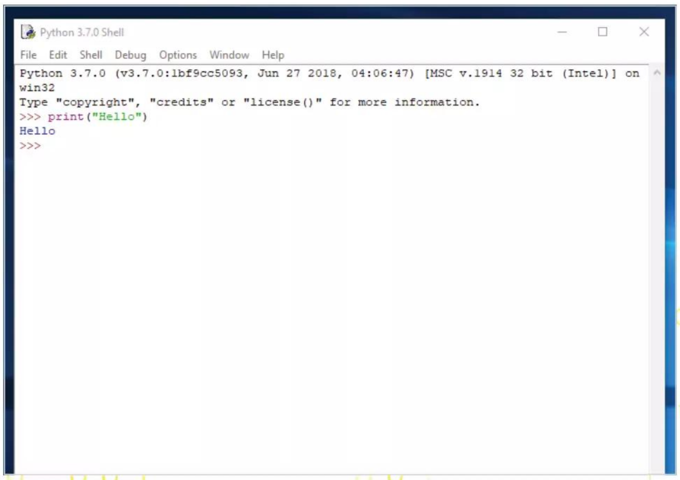
Essentially, you’ve already written your first piece of code with the print(“Hello everyone!”) function from the previous tutorial. However, let’s expand that and look at entering your code and playing around with some other Python examples.

PLAYING WITH PYTHON

As with most languages, computer or human, it’s all about remembering and applying the right words to the right situation. You’re not born knowing these words, so you need to learn them.

STEP 1 If you’ve closed Python 3 IDLE, re-open it as you did in the previous page. In the Shell, enter the familiar following:

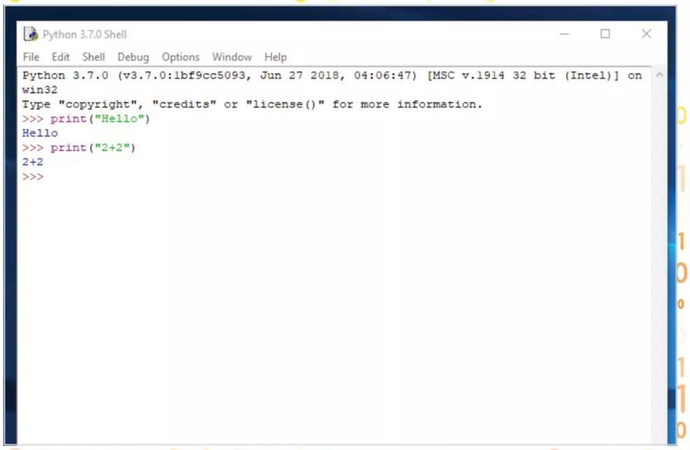
`print(“Hello”)`


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>>
```

STEP 2 As predicted, the word Hello appears in the Shell as blue text indicating output from a string. It’s fairly straightforward, and doesn’t require too much explanation. Now try:

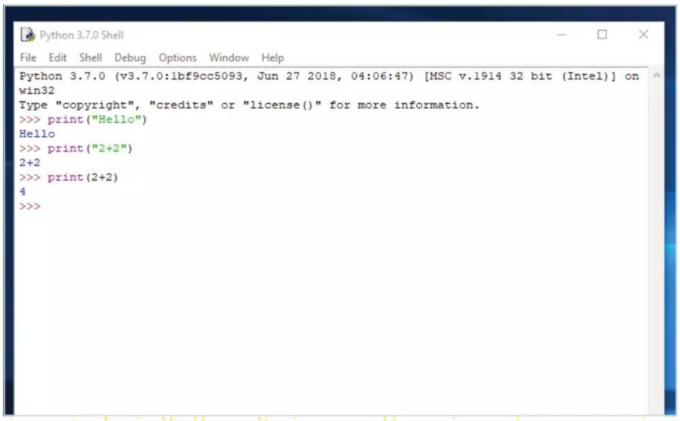
`print(“2+2”)`


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>>
```

STEP 3 You’ll notice that instead of the number 4, the output is the 2+2 you asked to be printed to the screen. The quotation marks are defining what’s being outputted to the IDLE Shell, to print the total of 2+2 you’ll need to remove the quotes:

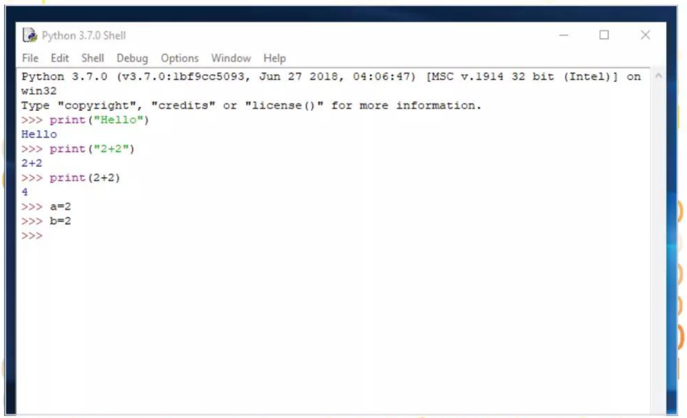
`print(2+2)`


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
```

STEP 4 You can continue as such, printing 2+2, 464+2343 and so on to the Shell. An easier way is to use a variable, which is something we will cover in more depth later. For now, enter:

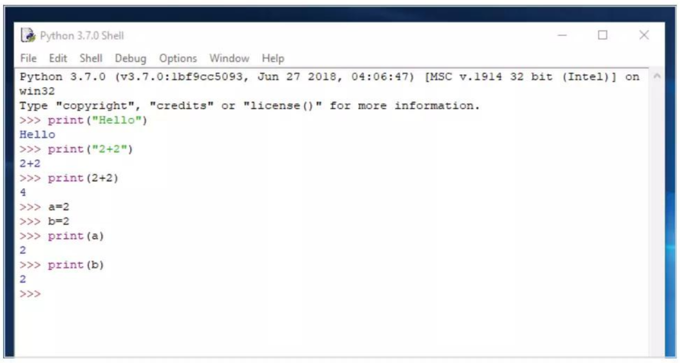
```
a=2
b=2
```


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>> a=2
>>> b=2
>>>
```

STEP 5 What you have done here is assign the letters a and b two values: 2 and 2. These are now variables, which can be called upon by Python to output, add, subtract, divide and so on, for as long as their numbers stay the same. Try this:

```
print(a)
print(b)
```

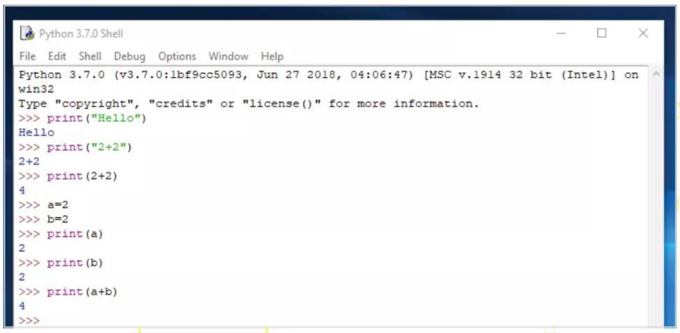

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>>
```

STEP 6 The output of the last step displays the current values of a and b individually, as essentially you've asked them to be printed separately. If you want to add them up, you can use the following:

```
print(a+b)
```

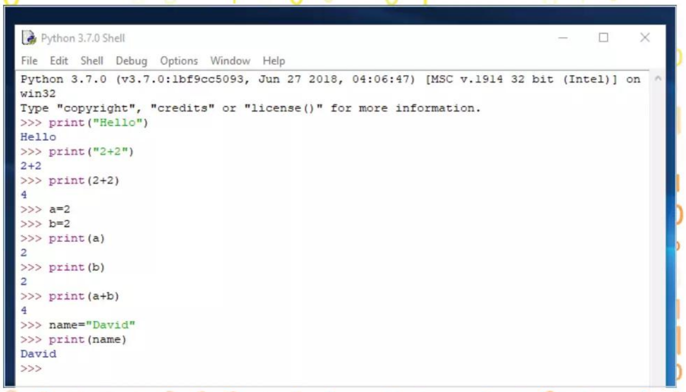
This code takes the value of both a and b, adds them together, and outputs the result.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>> print(a+b)
4
>>>
```

STEP 7 You can play around with different kinds of variables together with the Print function. For example, we could assign variables for someone's name:

```
name=”David”
print(name)
```

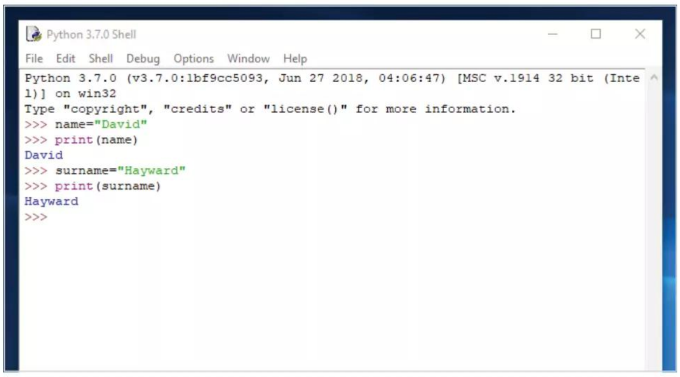

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>> print(a+b)
4
>>> name="David"
>>> print(name)
David
>>>
```

STEP 8 Now let's add a surname:

```
surname=”Hayward”
print(surname)
```

We now have two variables containing both a first name and a surname, and we can print them independently.

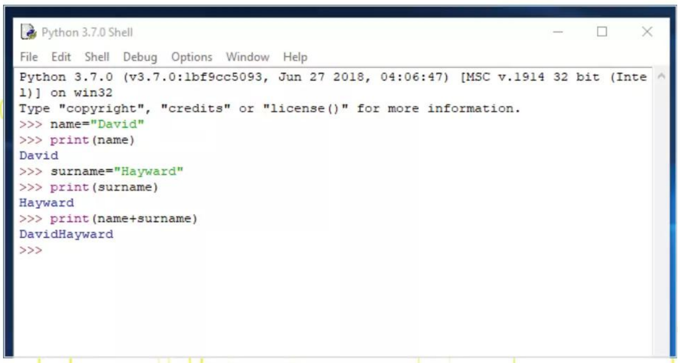

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>>
```

STEP 9 If we were to apply the same routine as before, using the + symbol, the name wouldn't appear correctly in the output in the Shell. Try it:

```
print(name+surname)
```

We need a space between the two, defining them as two separate values and not something you mathematically play around with.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> print(name+surname)
DavidHayward
>>>
```

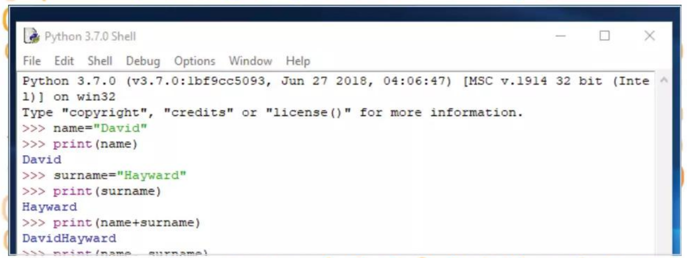
STEP 10 In Python 3 we can separate the two variables with a space by using a comma:

```
print(name, surname)
```

Alternatively, you can add the space yourself:

```
print(name+” “+surname)
```

As you can see, the use of the comma is much neater. Congratulations, you've just taken your first steps into the wide world of Python.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> print(name+surname)
DavidHayward
>>> print(name, surname)
```

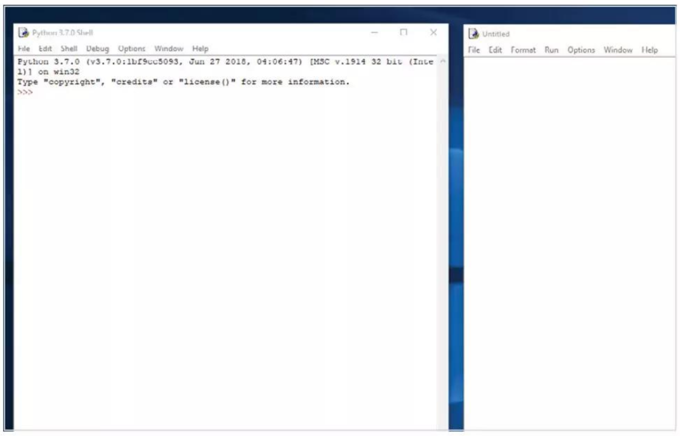
Saving and Executing Your Code

While working in the IDLE Shell is perfectly fine for snippets of code, it's not designed for entering longer program listings. In this section, we'll introduce you to the IDLE Editor, where most of our code will be entered from now on.

EDITING CODE

You will eventually reach a point where you have to move on from inputting single lines of code into the Shell. Instead, the IDLE Editor will allow you to save and execute your Python code.

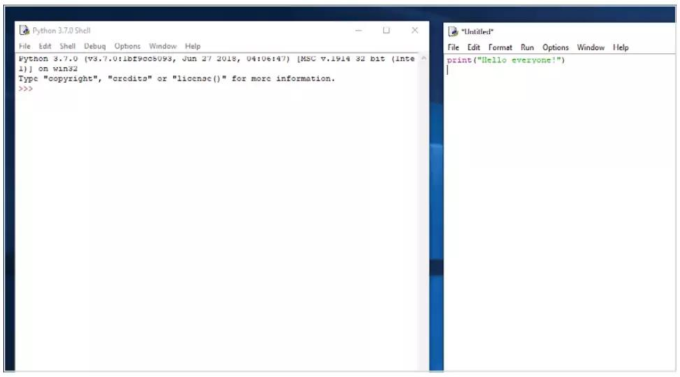
STEP 1 First, open the Python IDLE Shell. When it's up, click on **File > New File**, this will open a new window with Untitled as its name. This is the Python IDLE Editor, and within it, you can enter the code you need to create your future programs.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Untitled
File Edit Format Run Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

STEP 2 The IDLE Editor is, for all intents and purposes, a simple text editor with Python features, colour coding and so on. You enter code as you would within the Shell, so taking an example from the previous tutorial, enter:

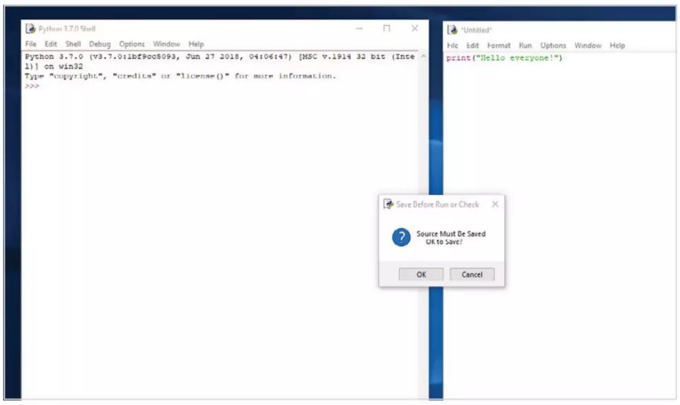
`print("Hello everyone!")`


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Untitled
File Edit Format Run Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
print("Hello everyone!")
```

STEP 3 As you can see the same colour coding is in place in the IDLE Editor as it is in the Shell, enabling you to better understand what's going on with your code. To execute the code, however, you need to first save it. Press **F5** and you'll have a Save…Check box open.

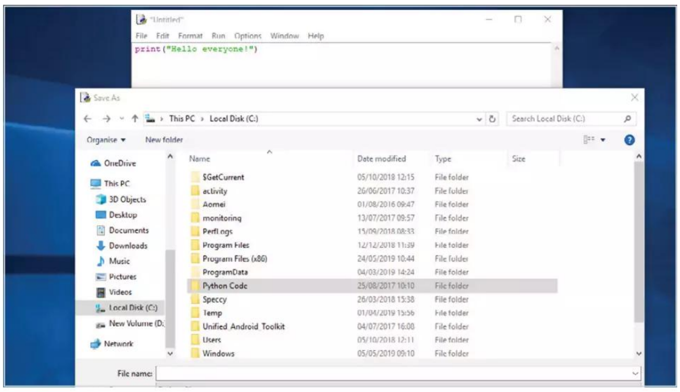

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Untitled
File Edit Format Run Options Window Help
Save Before Run or Check
OK
Cancel

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
print("Hello everyone!")
```

Source Must Be Saved
OK to Save?

STEP 4 Click on the **OK** button in the Save box, and select a destination where you'll save all your Python code. The destination can be a dedicated folder called Python, or you can just dump it wherever you like. Remember to keep a tidy file system, though, it'll help you out in the future.

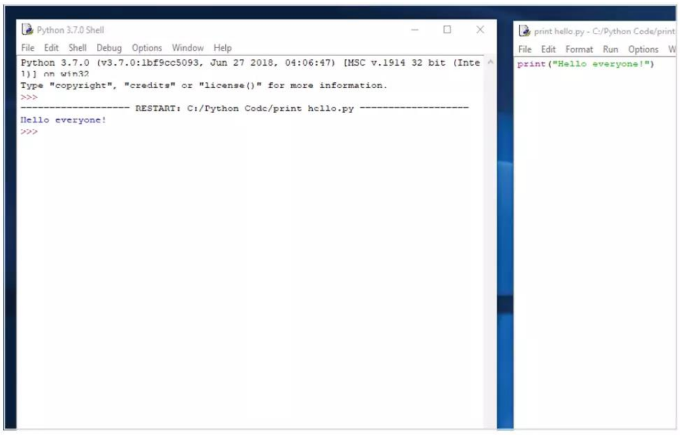

Untitled
File Edit Format Run Options Window Help
Save As
This PC › Local Disk (C:)
Search Local Disk (C:)
Organise ▾
New folder
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
New Volume (D:
Network
File name:

```
print("Hello everyone!")
```

Name	Date modified	Type	Size
$GetCurrent	05/10/2018 12:15	File folder	
activity	26/06/2017 10:37	File folder	
Aomei	01/08/2016 09:47	File folder	
monitoring	13/07/2017 09:57	File folder	
PerfLogs	15/09/2018 08:33	File folder	
Program Files	12/12/2018 11:39	File folder	
Program Files (x86)	24/05/2019 10:44	File folder	
ProgramData	04/03/2019 14:24	File folder	
Python Code	25/08/2017 10:10	File folder	
Speccy	26/03/2018 15:38	File folder	
Temp	01/04/2019 15:56	File folder	
Unified Android Toolkit	04/07/2017 16:08	File folder	
Users	05/10/2018 12:11	File folder	
Windows	05/05/2019 09:10	File folder	

STEP 5

Enter a name for your code, 'print hello' for example, and click on the Save button. As soon as the Python code is saved, it's executed and the output will be detailed in the IDLE Shell; In this case, the words 'Hello everyone!'.

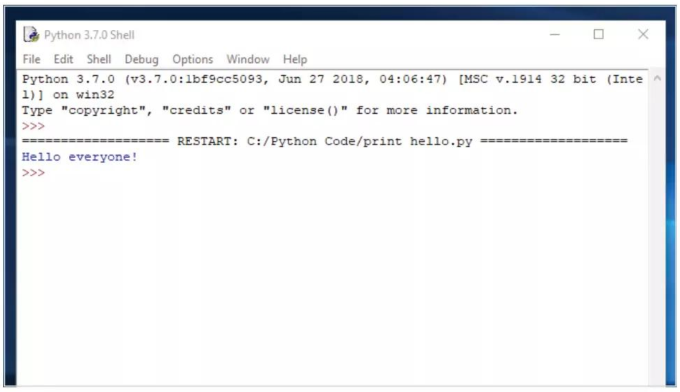

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
print hello.py - C:/Python Code/print
File Edit Format Run Options W

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
1)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
-------------------- RESTART: C:/Python Code/print hello.py --------------------
Hello everyone!
>>>
```

```
print("Hello everyone!")
```

STEP 6

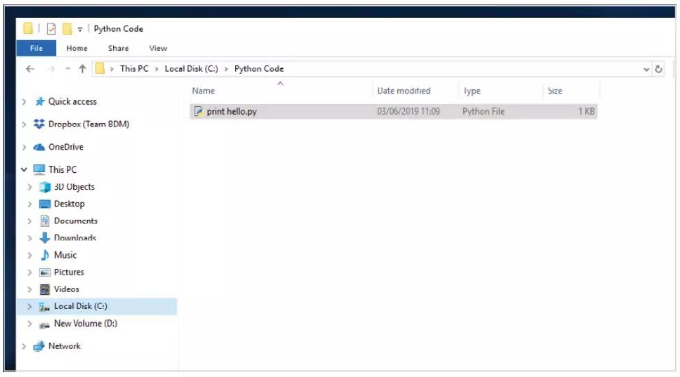
This is how the vast majority of your Python code will be conducted. Enter it into the Editor, hit F5, save the code, and look at the output in the Shell. Sometimes things will differ, depending on whether you've requested a separate window, but essentially that's the process and, unless otherwise stated, this is the method we will use.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
1)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Python Code/print hello.py ===================
Hello everyone!
>>>
```

STEP 7

If you open the file location of the saved Python code, you'll notice that it ends in a .py extension. This is the default Python filename, any code you create will be whatever.py, and any code downloaded from the many Internet Python resource sites will be .py. Just ensure that the code is written for Python 3.


Python Code
File
Home
Share
View
This PC › Local Disk (C:) › Python Code
Quick access
Dropbox (Team BDM)
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
New Volume (D:)
Network

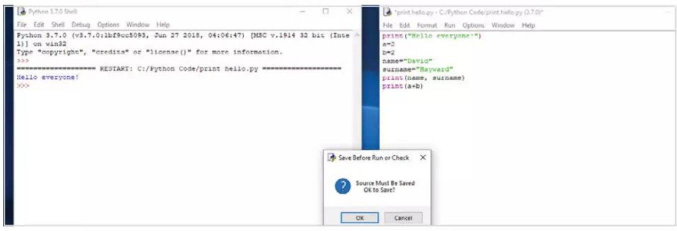
Name	Date modified	Type	Size
print hello.py	03/06/2019 11:09	Python File	1 KB

STEP 8

Let's extend the code and enter a few examples from the previous tutorial:

```
a=2
b=2
name="David"
surname="Hayward"
print(name, surname)
print (a+b)
```

If you press **F5** now, you'll be asked to save the file again, as it's been modified from before.

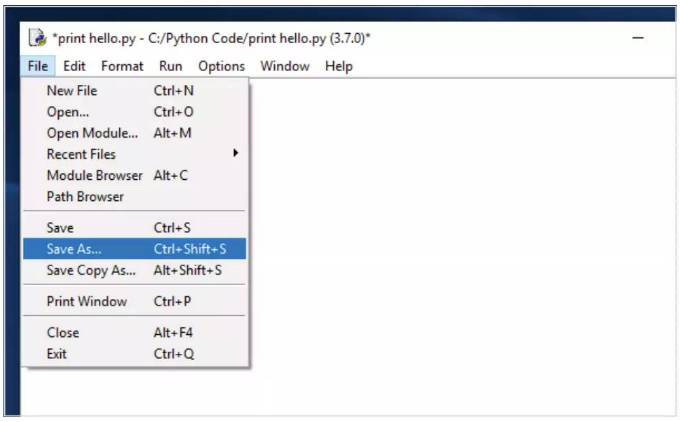

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
print hello.py - C:/Python Code/print hello.py (3.7.0)
File Edit Format Run Options Window Help
Save Before Run or Check
Source Must Be Saved
OK to Save?
OK
Cancel

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
1)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:/Python Code/print hello.py ===================
Hello everyone!
>>>
```

```
print("Hello everyone!")
a=2
b=2
name="David"
surname="Hayward"
print(name, surname)
print(a+b)
```

STEP 9

If you click the **OK** button the file will be overwritten with the new code entries, and executed; with the output in the Shell. It's not a problem with just these few lines, but if you were to edit a larger file overwriting can become an issue. Instead, use **File > Save As** from within the Editor to create a backup.

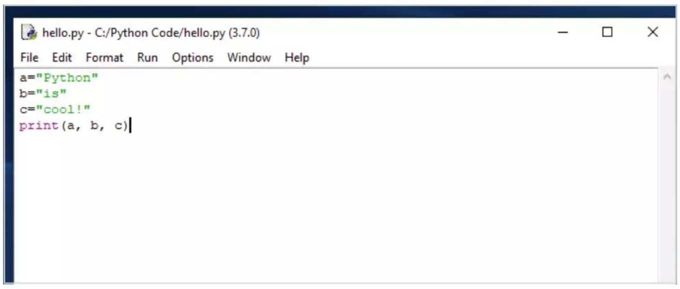

print hello.py - C:/Python Code/print hello.py (3.7.0)
File Edit Format Run Options Window Help
New File Ctrl+N
Open... Ctrl+O
Open Module... Alt+M
Recent Files
Module Browser Alt+C
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Alt+F4
Exit Ctrl+Q

STEP 10

Now create a new file. Close the Editor, and open a new instance (File > New File from the Shell). Enter the following, and save it as **hello.py**:

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

We will use this code in the next tutorial.


hello.py - C:/Python Code/hello.py (3.7.0)
File Edit Format Run Options Window Help

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

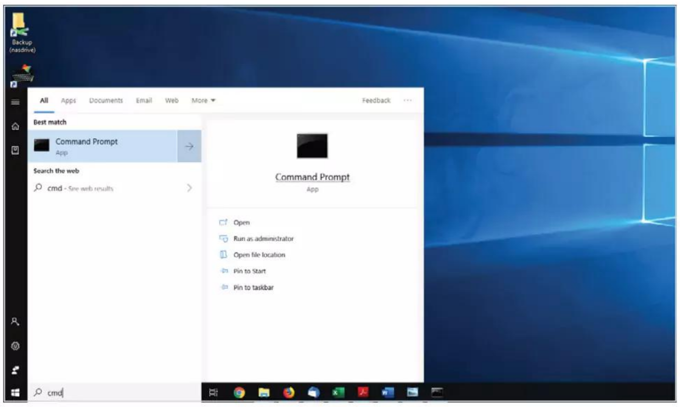
Executing Code from the Command Line

While we're going to be working from the GUI IDLE, it's worth taking a moment to look at Python's command line handling. Sometimes, depending on the code you write, executing via the command line is a better solution over the IDLE.

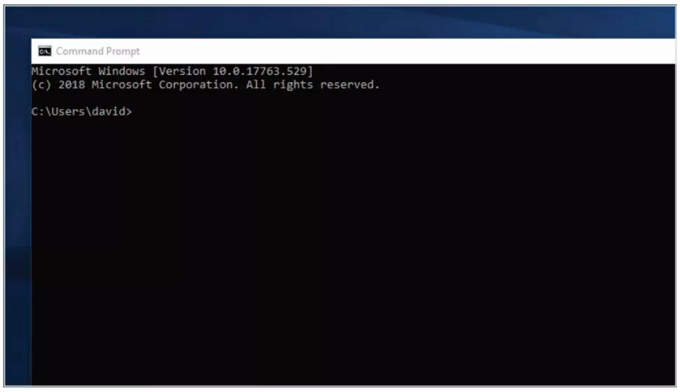
COMMAND THE CODE

Using the code we created in the previous tutorial, the one we named hello.py, let's see how we can run code that was made in the GUI at the command line level.

STEP 1 When you first installed Python, the installation routine automatically included all the necessary components to allow the execution of code outside of the GUI IDLE; in other words, the command line. To begin with, click on the Windows Start Button, and type: `cmd`.

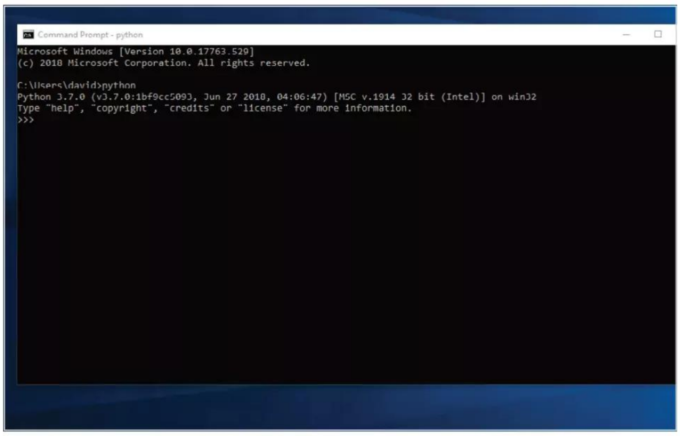

Backup (nasdrive)
All
Apps
Documents
Email
Web
More
Feedback
Best match
Command Prompt
App
Search the web
cmd - See web results
Command Prompt
App
Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
cmd

STEP 2 As you did when launching the Python IDLE, click on the returned result from the search, the Command Prompt App. This will launch a new window, with a black background and white text. This is the command line, also called a Terminal in macOS, Linux, and Raspberry Pi operating systems.


Command Prompt

```
Microsoft Windows [Version 10.0.17763.529]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\david>
```

STEP 3 Now you're at the command line, we can start Python using the command `python` and pressing the Enter key. This will put you into the command line version of the Shell, with the familiar, three right-facing arrows as the cursor (>>>).

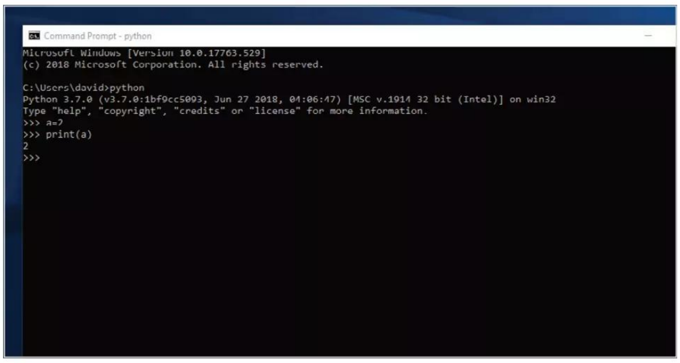

Command Prompt - python

```
Microsoft Windows [Version 10.0.17763.529]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\david>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

STEP 4 From here you're able to enter the code you've looked at previously, such as:

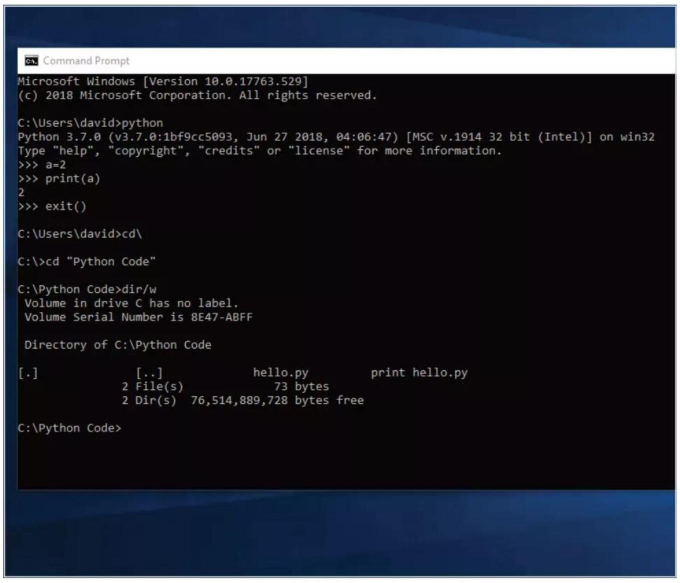
```
a=2
print(a)
```

As you can see, it works exactly the same.


Command Prompt - python

```
Microsoft Windows [Version 10.0.17763.529]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\david>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>>
```

STEP 5 Now enter `exit()` to leave the command line Python session, and return back to the command prompt. Enter the folder where you saved the code from the previous tutorial, and list the available files within; you should see the **hello.py** file.



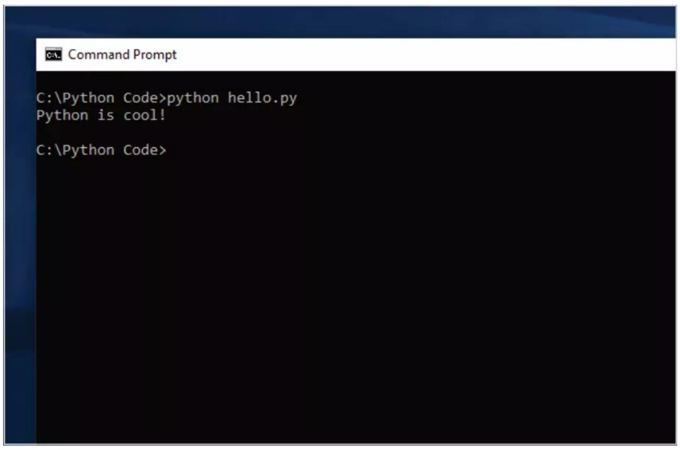
```
Microsoft Windows [Version 10.0.17763.529]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\david>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> exit()

C:\Users\david>cd\

C:\>cd "Python Code"

C:\Python Code>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of C:\Python Code

[.]              [..]             hello.py         print hello.py
               2 File(s)             73 bytes
               2 Dir(s)  76,514,889,728 bytes free

C:\Python Code>
```

STEP 6 From within the same folder as the code you're going to run, enter the following into the command line:

```
python hello.py
```

This will execute the code we created, which to remind you is:

```
a=”Python”
b=”is”
c=”cool!”
print(a, b, c)
```



```
C:\Python Code>python hello.py
Python is cool!

C:\Python Code>
```

DIFFERENT VERSIONS OF PYTHON

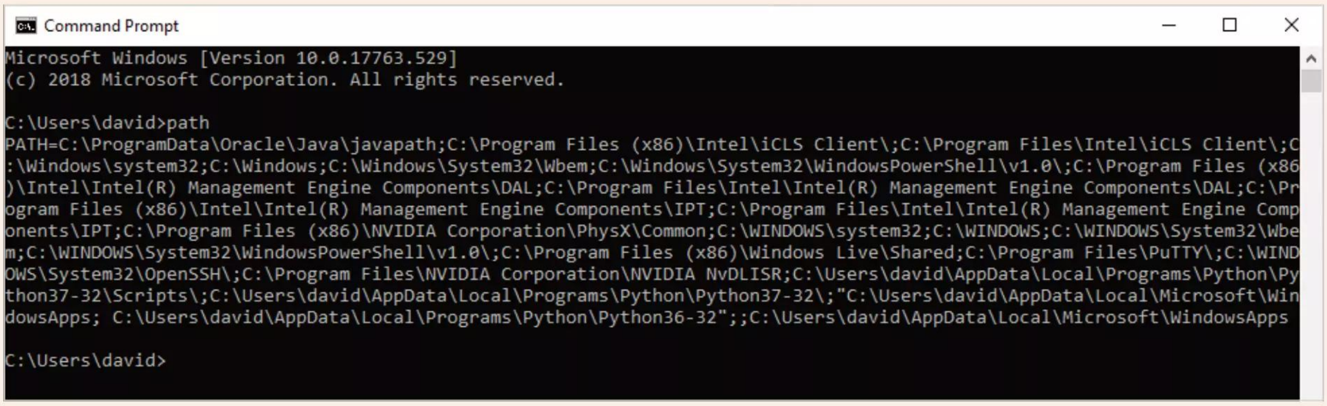
If you've previously used Python 3 on a Mac or Linux, and subsequently the Raspberry Pi, you may be a little confused as to why the Windows version of Python uses the command line: **python**, instead of **python3**.

The reason behind this is that UNIX-like systems, such as macOS and Linux, already have Python libraries pre-installed. These older libraries are present because some of the macOS and Linux system utilities rely on Python 2, and therefore installing a newer version of Python, and thus altering the executable name, could have dire consequences to the system.

As a result, developers decided that the best approach for macOS and Linux systems would be to leave the command line **'python'** as exclusive Python 2 use, and newer versions of user-installed Python would be **'python3'**.

This isn't an issue with Windows, as it doesn't use any Python libraries other than the ones installed by the user themselves when actually installing Python. When a Windows user installs Python, the installation wizard will auto-include the command line instance to the core Windows PATH variable, which you can view by entering: `path` into the command line. This points to the python.exe file required to execute Python code from the command line.

We don't recommend you install both Python 2 and Python 3 within Windows 10; naturally, you can if you want, but realistically, although Python 2 still has a foothold in the coding world, Python 3 is the newest version. However, if you do, then you will need to rename one of the Python versions names; as they will be installed in different folders and both use python.exe as the command line executable. It's a little long-winded, so unless there's a dire need to have both versions of Python installed, it's best to stick to Python 3.



```
Microsoft Windows [Version 10.0.17763.529]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\david>path
PATH=C:\ProgramData\Oracle\Java\javapath;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C
:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86
)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Pr
ogram Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files\Intel\Intel(R) Management Engine Comp
onents\IPT;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbe
m;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Windows Live\Shared;C:\Program Files\PuTTY\;C:\WIND
OWS\System32\OpenSSH\;C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR;C:\Users\david\AppData\Local\Programs\Python\Py
thon37-32\Scripts\;C:\Users\david\AppData\Local\Programs\Python\Python37-32\;"C:\Users\david\AppData\Local\Microsoft\Win
dowsApps; C:\Users\david\AppData\Local\Programs\Python\Python36-32";;C:\Users\david\AppData\Local\Microsoft\WindowsApps

C:\Users\david>
```

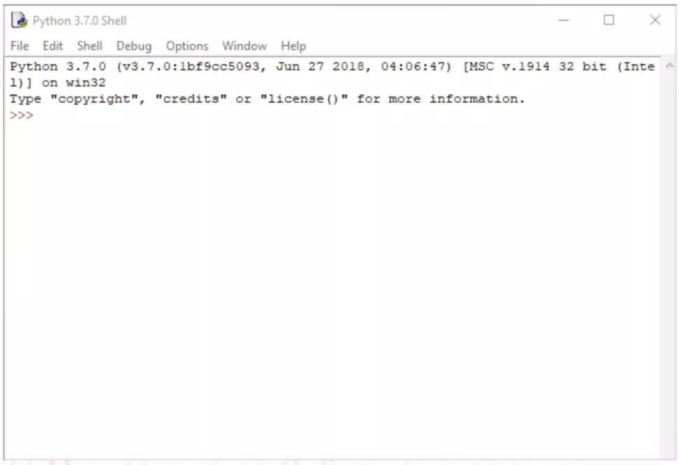
Numbers and Expressions

We've seen some basic mathematical expressions with Python, simple addition and the like. Now let's expand on that, and see just how powerful Python is as a calculator. You can work within the IDLE Shell, or in the Editor, whichever you like.

IT'S ALL MATHS, MAN

You can get some really impressive results from the mathematical powers of Python, as maths is the driving force behind the code with most, if not all, programming languages.

STEP 1 Open up the GUI version of Python 3, as mentioned you can use either the Shell or the Editor. For the time being, we're going to use the Shell. If you've opted to use a third-party text editor, note that you need to get to the IDLE Shell for this part of the tutorial.

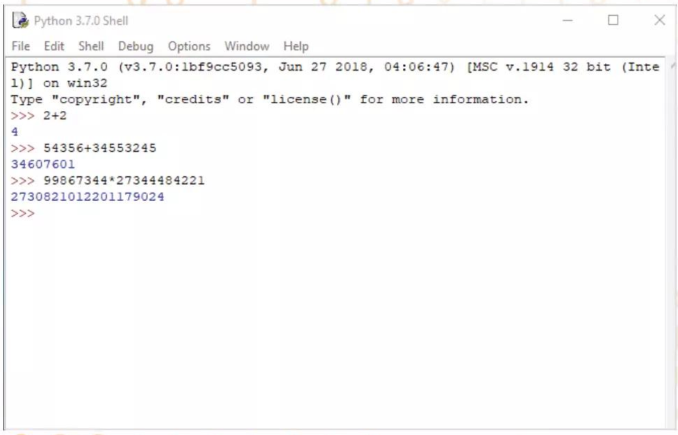

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

STEP 2 In the Shell enter the following:

```
2+2
54356+34553245
99867344*27344484221
```

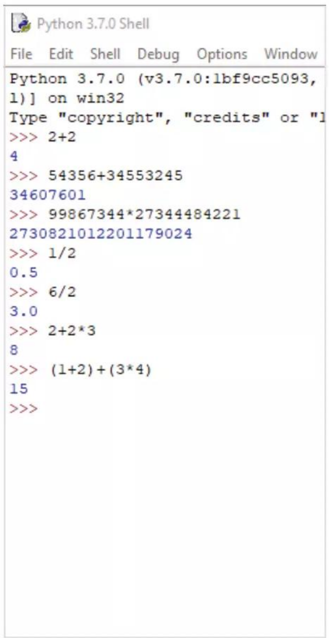
As you can see, Python can handle some quite large numbers.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>>
```

STEP 3 You can use all the customary Mathematical operations: divide, multiply, brackets and so on. Practise with a few, for example:

```
1/2
6/2
2+2*3
(1+2)+(3*4)
```

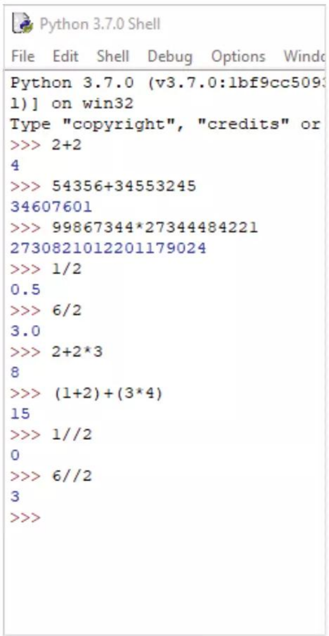

Python 3.7.0 Shell
File Edit Shell Debug Options Window

```
Python 3.7.0 (v3.7.0:1bf9cc5093,
1)] on win32
Type "copyright", "credits" or "
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>>
```

STEP 4 As you've no doubt noticed, division produces a decimal number. In Python, these are called floats, or floating point arithmetic. If however, you need an integer as opposed to a decimal answer, then you can use a double slash:

```
1//2
6//2
```

and so on.


Python 3.7.0 Shell
File Edit Shell Debug Options Windo

```
Python 3.7.0 (v3.7.0:1bf9cc509
1)] on win32
Type "copyright", "credits" or
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>>
```

STEP 5

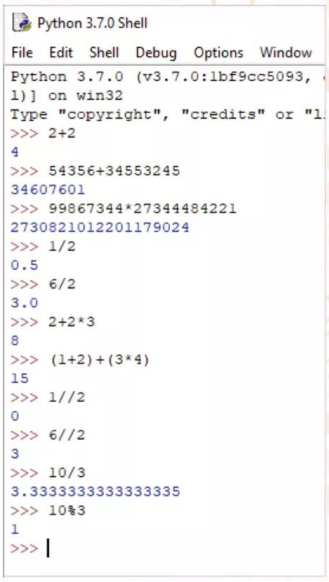
You can also use an operation to see the remainder left over from division. For example:

`10/3`

will display 3.333333333, which is, of course, 3.3-recurring. If you now enter:

`10%3`

This will display 1, which is the remainder left over from dividing 10 by 3.



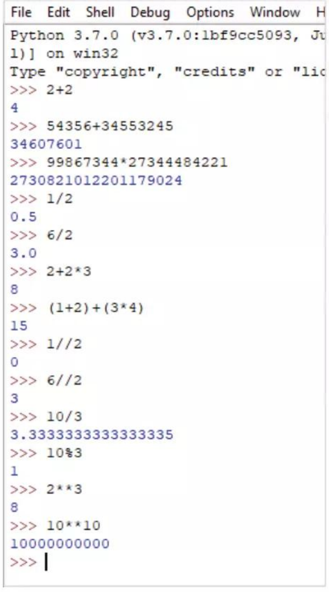
```
Python 3.7.0 (v3.7.0:1bf9cc5093,
1)] on win32
Type "copyright", "credits" or "l
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 
```

STEP 6

Next up we have the power operator, or exponentiation if you want to be technical. To work out the power of something you can use a double multiplication symbol, or double-star on the keyboard:

```
2**3
10**10
```

Essentially, it's 2x2x2, but we're sure you already know the basics behind maths operators. This is how you would work it out in Python.

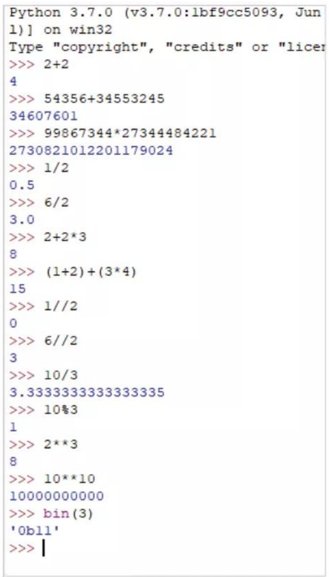


```
Python 3.7.0 (v3.7.0:1bf9cc5093, Ju
1)] on win32
Type "copyright", "credits" or "lic
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> 
```

STEP 7

Numbers and expressions don't stop there. Python has numerous built-in functions to work out sets of numbers, absolute values, complex numbers, and a host of Mathematical expressions and Pythogarian tongue-twisters. For example, to convert a number to binary, use:

`bin(3)`



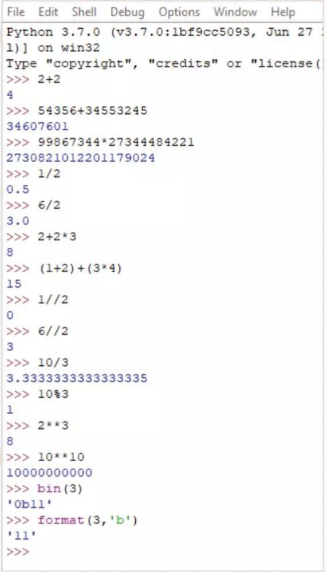
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun
1)] on win32
Type "copyright", "credits" or "licen
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> 
```

STEP 8

This will be displayed as '0b11', converting the integer into binary, and adding the prefix 0b to the front. If you want to remove the 0b prefix, then you can use:

`format(3, 'b')`

The Format command converts a value, the number 3, to a formatted representation as controlled by the format specification, the 'b' part.

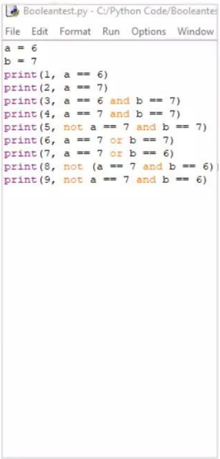


```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27
1)] on win32
Type "copyright", "credits" or "license(
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> format(3,'b')
'11'
>>>
```

STEP 9

A Boolean Expression is a logical statement that will either be true or false. We can use these to compare data, and test to see if it's equal to, less than, or greater than. Try this in a New File:

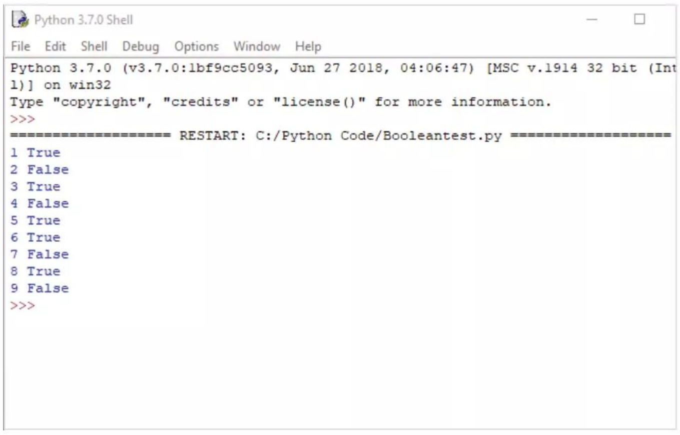
```
a = 6
b = 7
print(1, a == 6)
print(2, a == 7)
print(3, a == 6 and b == 7)
print(4, a == 7 and b == 7)
print(5, not a == 7 and b == 7)
print(6, a == 7 or b == 7)
print(7, a == 7 or b == 6)
print(8, not (a == 7 and b == 6))
print(9, not a == 7 and b == 6)
```


Booleantest.py - C:/Python Code/Booleante
File Edit Format Run Options Window

```
a = 6
b = 7
print(1, a == 6)
print(2, a == 7)
print(3, a == 6 and b == 7)
print(4, a == 7 and b == 7)
print(5, not a == 7 and b == 7)
print(6, a == 7 or b == 7)
print(7, a == 7 or b == 6)
print(8, not (a == 7 and b == 6)
print(9, not a == 7 and b == 6)
```

STEP 10

Execute the code from Step 9, and you'll see a series of True or False statements depending on the result of the two defining values: 6 and 7. It's an extension of what we've looked at, and an important part of programming.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Int
1)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Python Code/Booleantest.py ====================
1 True
2 False
3 True
4 False
5 True
6 True
7 False
8 True
9 False
>>>
```

Using Comments

When writing your code, the flow, what each variable does, how the overall program will operate and so on, is all inside your head. Another programmer could follow the code line by line, but when the code starts to hit thousands of lines, things get a little difficult to read.

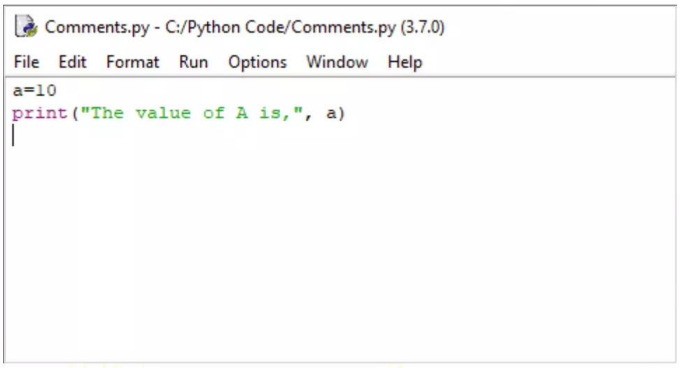
#COMMENTS!

A method used by most programmers for keeping their code readable, is by commenting on certain sections. For example, if a variable is used, the programmer comments on what it's supposed to do. It's just good practise.

STEP 1 We'll start by creating a new instance of the IDLE Editor (**File > New File**), and then create a simple variable and print command:

```
a=10
print(“The value of A is,”, a)
```

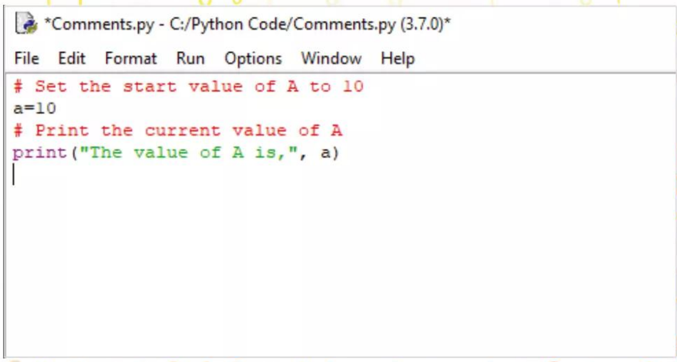
Save the file, and execute the code.


Comments.py - C:/Python Code/Comments.py (3.7.0)
File Edit Format Run Options Window Help

```
a=10
print("The value of A is,", a)
```

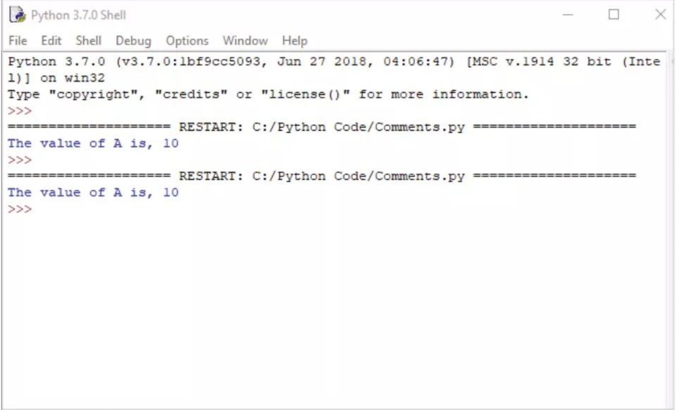
STEP 2 Running the code will return the line: **The value of A is, 10** into the IDLE Shell window – which is what we expected. Now let's add some of the types of comments you'd normally see within code:

```
# Set the start value of A to 10
a=10
# Print the current value of A
print(“The value of A is,”, a)
```


Comments.py - C:/Python Code/Comments.py (3.7.0)
File Edit Format Run Options Window Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
```

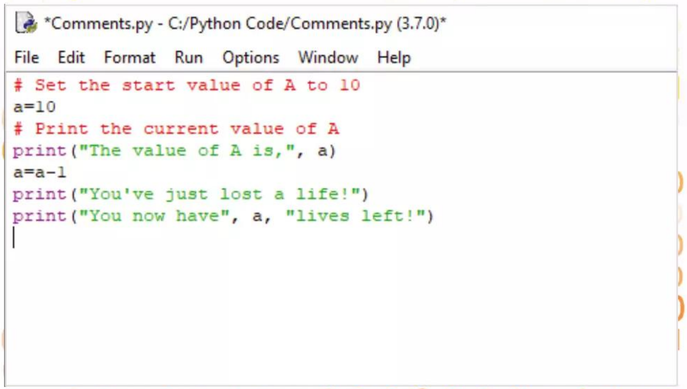
STEP 3 Re-save the code and execute it. You'll see that the output in the IDLE Shell is still the same as before, despite the extra lines being added. Simply put, the hash symbol (#) denotes a line of text the programmer can insert, to inform them and others of what's going on, without the user being aware.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Python Code/Comments.py ====================
The value of A is, 10
>>>
==================== RESTART: C:/Python Code/Comments.py ====================
The value of A is, 10
>>>
```

STEP 4 Let's assume that the variable A we've created is the number of lives in a game. Every time the player dies, the value decreases by 1. The programmer could insert a routine along the lines of:

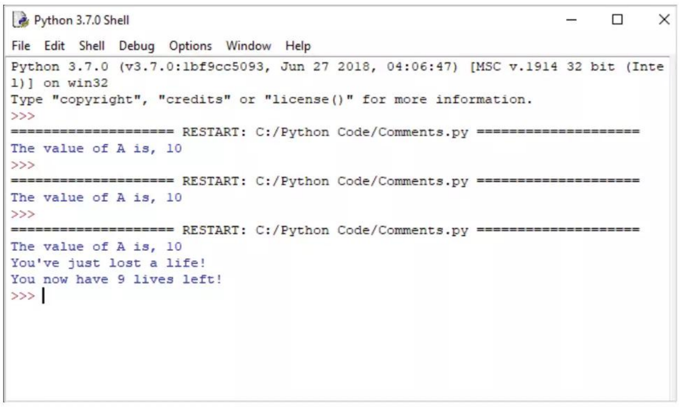
```
a=a-1
print(“You’ve just lost a life!”)
print(“You now have”, a, “lives left!”)
```


Comments.py - C:/Python Code/Comments.py (3.7.0)
File Edit Format Run Options Window Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
a=a-1
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 5

While we know that the variable A denotes number of lives and the player has just lost one, a casual viewer, or someone checking the code, may not know. Imagine for a moment that the code is twenty thousand lines long, instead of just our seven. You can see how handy comments are.

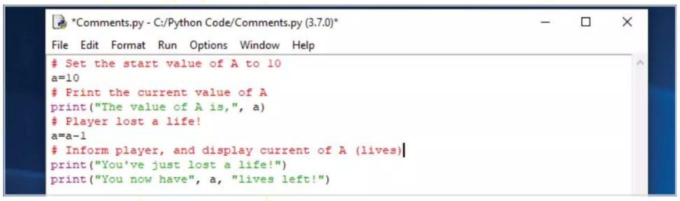

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Python Code/Comments.py ====================
The value of A is, 10
>>>
==================== RESTART: C:/Python Code/Comments.py ====================
The value of A is, 10
>>>
==================== RESTART: C:/Python Code/Comments.py ====================
The value of A is, 10
You've just lost a life!
You now have 9 lives left!
>>> 
```

STEP 6

Essentially, the new code together with comments could look like:

```
# Set the start value of A to 10
a=10
# Print the current value of A
print(“The value of A is,”, a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A
(lives)
print(“You’ve just lost a life!”)
print(“You now have”, a, “lives left!”)
```

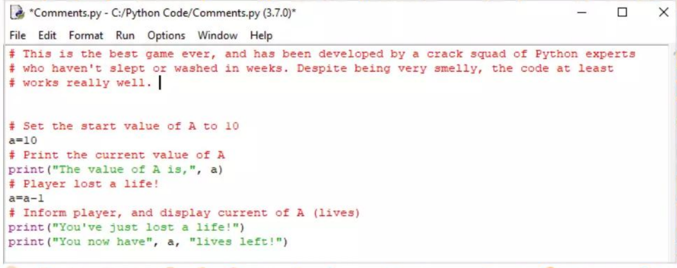

Comments.py - C:/Python Code/Comments.py (3.7.0)
File Edit Format Run Options Window Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 7

You can use comments in different ways. For example, Block Comments are a large section of text that details what’s going on in the code, such as telling the code reader which variables you’re planning on using:

```
# This is the best game ever, and has been
developed by a crack squad of Python experts
# who haven’t slept or washed in weeks. Despite
being very smelly, the code at least
# works really well.
```

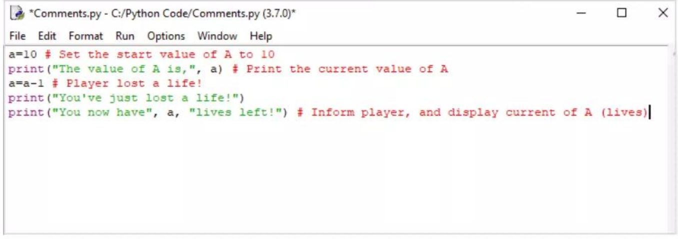

Comments.py - C:/Python Code/Comments.py (3.7.0)
File Edit Format Run Options Window Help

```
# This is the best game ever, and has been developed by a crack squad of Python experts
# who haven't slept or washed in weeks. Despite being very smelly, the code at least
# works really well. 


# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 8

Inline Comments are comments that follow a section of code. Take our examples from above, instead of inserting the code on a separate line, we could use:

```
a=10 # Set the start value of A to 10
print(“The value of A is,”, a) # Print the current
value of A
a=a-1 # Player lost a life!
print(“You’ve just lost a life!”)
print(“You now have”, a, “lives left!”) # Inform
player, and display current value of A (lives)
```

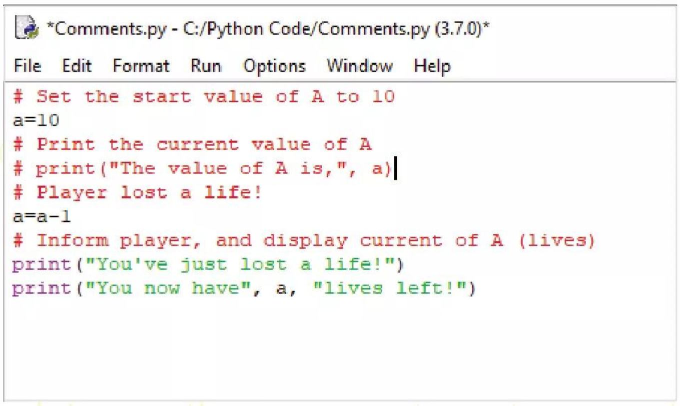

Comments.py - C:/Python Code/Comments.py (3.7.0)
File Edit Format Run Options Window Help

```
a=10 # Set the start value of A to 10
print("The value of A is,", a) # Print the current value of A
a=a-1 # Player lost a life!
print("You've just lost a life!")
print("You now have", a, "lives left!") # Inform player, and display current of A (lives)
```

STEP 9

The comment, the hash symbol, can also be used to comment out sections of code you don’t want to be executed in your program. For instance, if you wanted to remove the first print statement, you would use:

```
# print(“The value of A is,”, a)
```

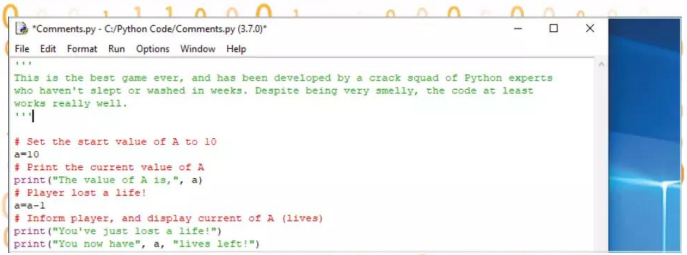

Comments.py - C:/Python Code/Comments.py (3.7.0)
File Edit Format Run Options Window Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
# print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 10

You also use three single quotes to comment out a Block Comment, or multi-line section of comments. For them to work, place them before and after the areas you want to comment:

```
‘’’
This is the best game ever, and has been developed
by a crack squad of Python experts
who haven’t slept or washed in weeks. Despite
being very smelly, the code at least
works really well.   ’’’
```


Comments.py - C:/Python Code/Comments.py (3.7.0)
File Edit Format Run Options Window Help

```
'''
This is the best game ever, and has been developed by a crack squad of Python experts
who haven't slept or washed in weeks. Despite being very smelly, the code at least
works really well.
'''

# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

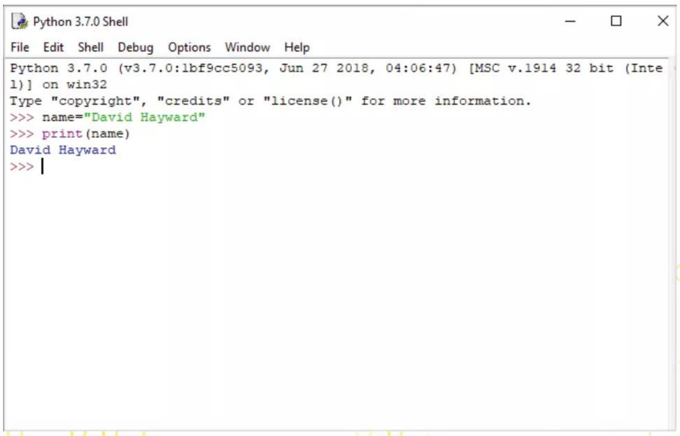
Working with Variables

We've seen some examples of variables in our Python code already, but it's always worth going through the way they operate, and how Python creates and assigns certain values to a variable.

VARIOUS VARIABLES

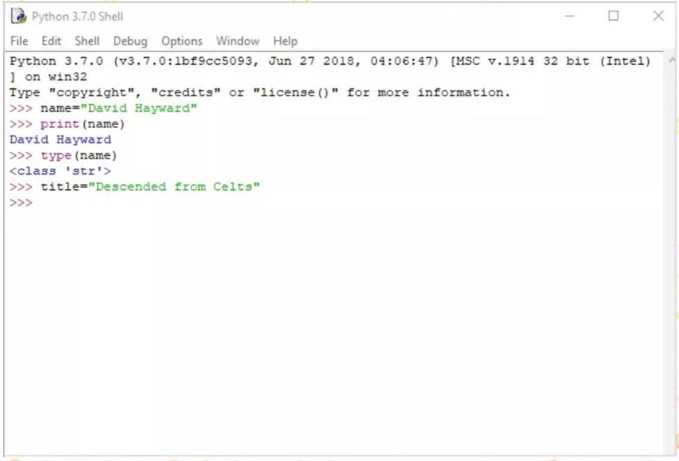
We'll be working with the Python 3 IDLE Shell in this tutorial. If you haven't already, open Python 3 or close down the previous IDLE Shell to clear up any old code.

STEP 1 In some programming languages, you're required to use a dollar sign to denote a string, which is a variable made up of multiple characters, such as a name of a person. In Python this isn't necessary, so, for example, in the Shell enter: `name="David Hayward"` (use your own name, unless you're also called David Hayward).



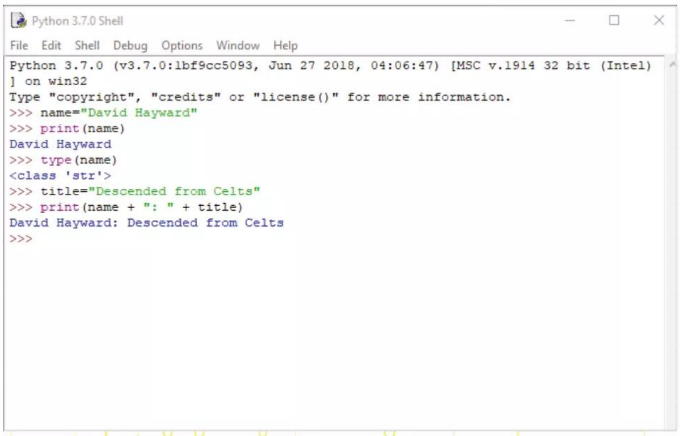
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print(name)
David Hayward
>>> |
```

STEP 2 You can check the type of variable in use by issuing the `type ()` command, placing the name of the variable inside the brackets. In our example, this would be: `type (name)`. Add a new string variable: `title="Descended from Celts"`.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)
] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print(name)
David Hayward
>>> type(name)
<class 'str'>
>>> title="Descended from Celts"
>>>
```

STEP 3 As we've seen previously, variables can be concatenated using the plus symbol between the variable names. In our example, we can use: `print (name + ": " + title)`. The middle part, between the quotations, allows us to add a colon and a space. As variables are connected without spaces, we need to add them manually.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)
] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print(name)
David Hayward
>>> type(name)
<class 'str'>
>>> title="Descended from Celts"
>>> print(name + ": " + title)
David Hayward: Descended from Celts
>>>
```

STEP 4 We can also combine variables within another variable. For example, to combine both name and title variables into a new variable, we use:

```
character=name + ": " +
title
```

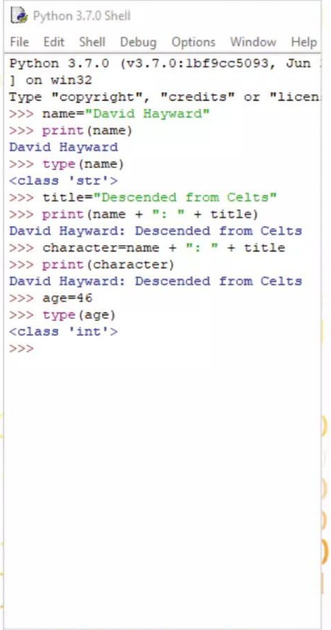
Then output the content of the new variable as:

```
print (character)
```

Numbers are stored as different variables:

```
age=44
Type (age)
```

Which, as we know, are integers.

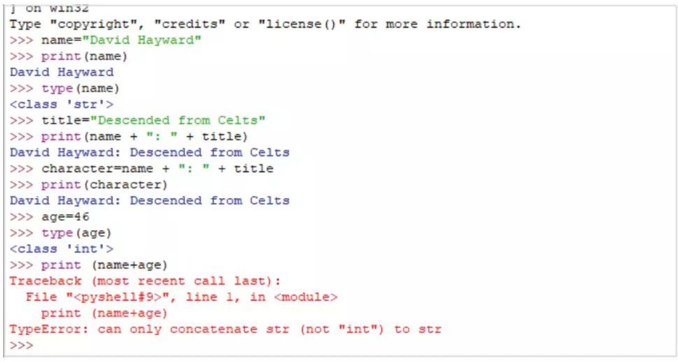


```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun
] on win32
Type "copyright", "credits" or "licen
>>> name="David Hayward"
>>> print(name)
David Hayward
>>> type(name)
<class 'str'>
>>> title="Descended from Celts"
>>> print(name + ": " + title)
David Hayward: Descended from Celts
>>> character=name + ": " + title
>>> print(character)
David Hayward: Descended from Celts
>>> age=46
>>> type(age)
<class 'int'>
>>>
```

STEP 5

However, you can't combine both strings and integer type variables in the same command as you would a set of similar variables. You'll need to turn one into the other, or vice versa. When you do try to combine both, you'll get an error message:

`print (name + age)`



```
] on win32
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print(name)
David Hayward
>>> type(name)
<class 'str'>
>>> title="Descended from Celts"
>>> print(name + ": " + title)
David Hayward: Descended from Celts
>>> character=name + ": " + title
>>> print(character)
David Hayward: Descended from Celts
>>> age=46
>>> type(age)
<class 'int'>
>>> print (name+age)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print (name+age)
TypeError: can only concatenate str (not "int") to str
>>>
```

STEP 6

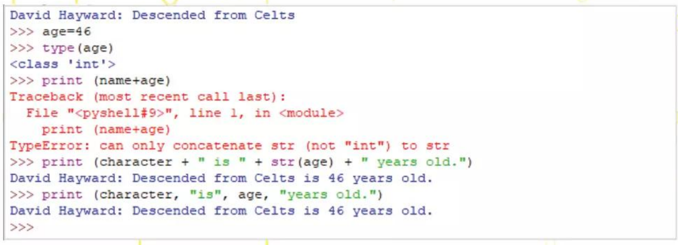
This is a process known as TypeCasting. The Python code is:

`print (character + " is " + str(age) + " years old.")`

Alternatively, you can use:

`print (character, "is", age, "years old.")`

Notice again that in the last example, you don't need the spaces between the words in quotes, as the commas treat each argument to print separately.



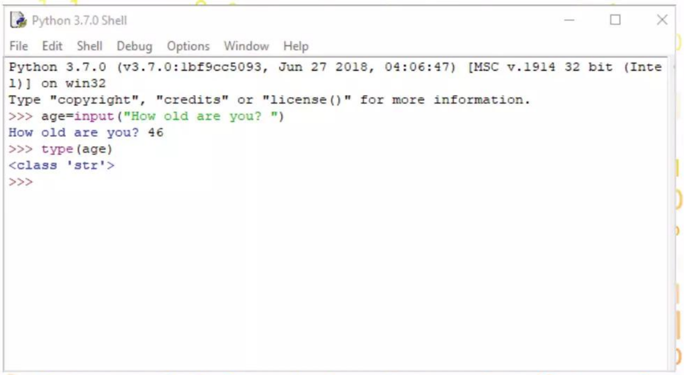
```
David Hayward: Descended from Celts
>>> age=46
>>> type(age)
<class 'int'>
>>> print (name+age)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print (name+age)
TypeError: can only concatenate str (not "int") to str
>>> print (character + " is " + str(age) + " years old.")
David Hayward: Descended from Celts is 46 years old.
>>> print (character, "is", age, "years old.")
David Hayward: Descended from Celts is 46 years old.
>>>
```

STEP 7

Another example of TypeCasting is when you ask for input from the user, such as a number. for example, enter:

`age= input ("How old are you? ")`

All data stored from the Input command is stored as a string variable.



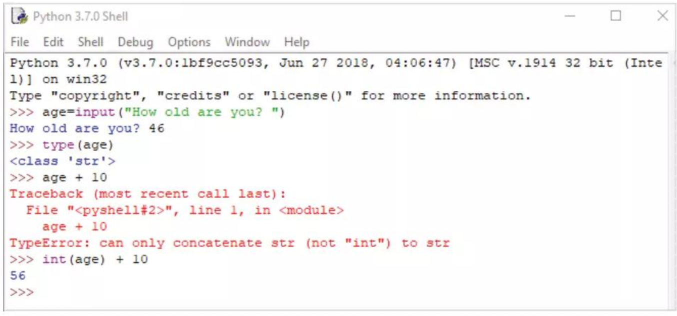
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> age=input("How old are you? ")
How old are you? 46
>>> type(age)
<class 'str'>
>>>
```

STEP 8

This presents a bit of a problem when you want to work with a number that's been inputted by the user, for example, as age + 10 is both a string variable and an integer, it won't work. Instead, you need to enter:

`int(age) + 10`

This will TypeCast the age string into an integer that can be worked with.



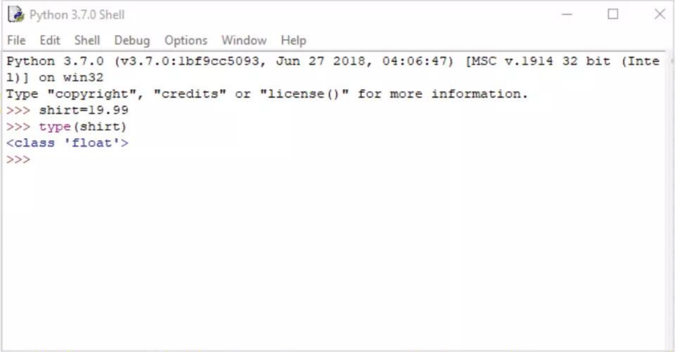
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> age=input("How old are you? ")
How old are you? 46
>>> type(age)
<class 'str'>
>>> age + 10
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    age + 10
TypeError: can only concatenate str (not "int") to str
>>> int(age) + 10
56
>>>
```

STEP 9

The use of TypeCasting is also important when dealing with floating point arithmetic; remember: numbers that have a decimal point in them. For example, enter:

`shirt=19.99`

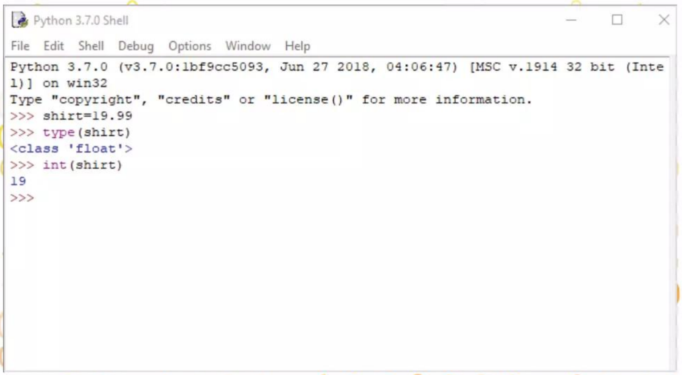
Now enter: `type(shirt)` and you'll see that Python has allocated the number as a 'float', because the value contains a decimal point.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>
>>>
```

STEP 10

When combining integers and floats Python usually converts the integer to a float, but should the reverse ever be applied, it's worth remembering that Python doesn't return the exact value. When converting a float to an integer, Python will always round down to the nearest integer, called truncating; in our case instead of 19.99, it becomes 19.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>
>>> int(shirt)
19
>>>
```

User Input

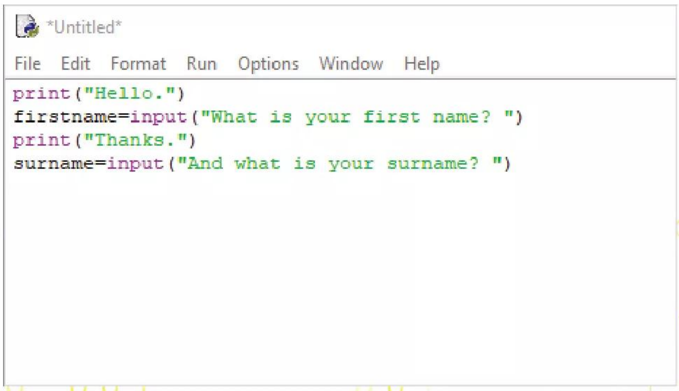
We've seen some basic user interaction with the code from a few of the examples earlier, so now would be a good time to focus solely on how you get information from the user, then store and present it.

USER FRIENDLY

The type of input you want from the user will depend greatly on the type of program you're coding. A game, for example, may ask for a character's name, whereas a database can ask for personal details.

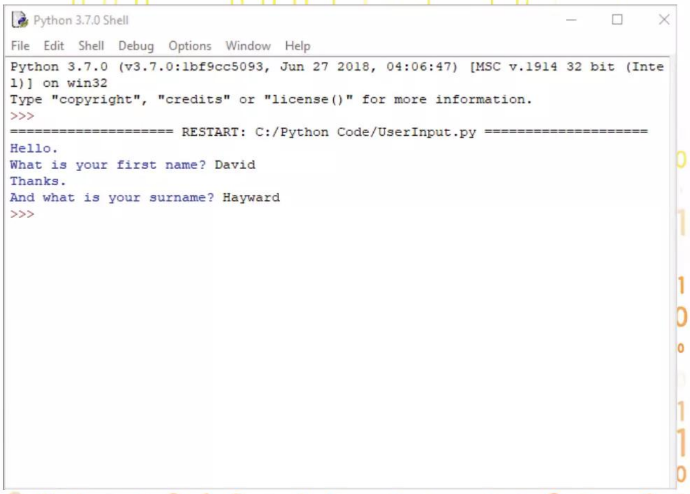
STEP 1 If it's not already, open the Python 3 IDLE Shell, and start a New File in the Editor. Let's begin with something really simple, enter:

```
print(“Hello”)
firstname=input(“What is your first name? ”)
print(“Thanks.”)
surname=input(“And what is your surname? ”)
```


Untitled
File Edit Format Run Options Window Help

```
print("Hello.")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
```

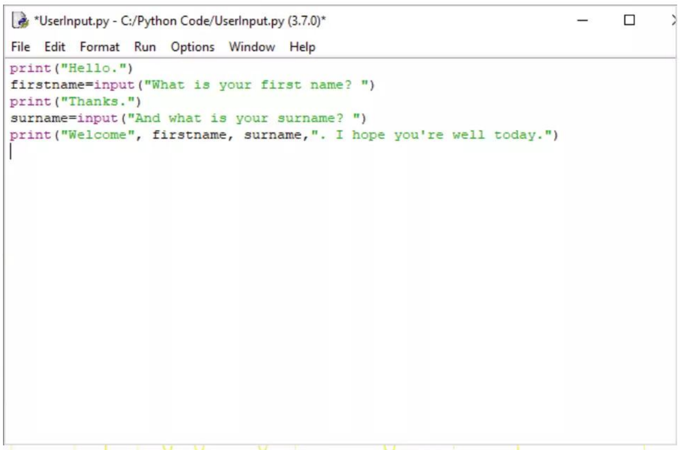
STEP 2 Save and execute the code, and, as you no doubt suspected, in the IDLE Shell the program will ask for your first name, storing it as the variable **firstname**, followed by your surname; also stored in its own variable **(surname)**.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Python Code/UserInput.py ====================
Hello.
What is your first name? David
Thanks.
And what is your surname? Hayward
>>>
```

STEP 3 Now that we have the user's name stored in a couple of variables, we can call them up whenever we want:

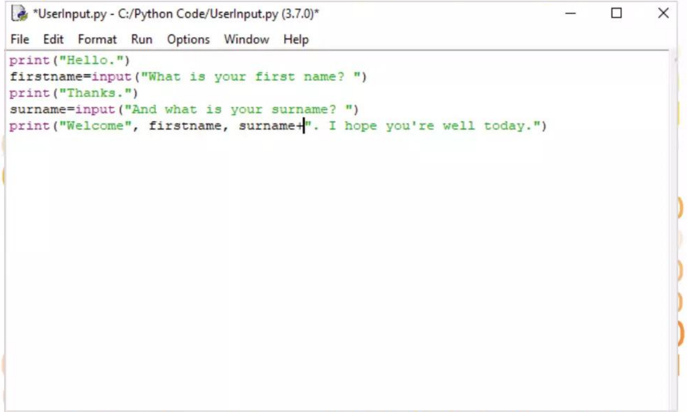
```
print(“Welcome”, firstname, surname, “. I hope
you’re well today.”)
```


UserInput.py - C:/Python Code/UserInput.py (3.7.0)
File Edit Format Run Options Window Help

```
print("Hello.")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
print("Welcome", firstname, surname,". I hope you're well today.")
```

STEP 4 Run the code and you'll notice a slight issue, the full stop after the surname follows a blank space. To eliminate that, we can add a plus sign instead of the comma in the code:

```
print(“Welcome”, firstname, surname+“. I hope
you’re well today.”)
```

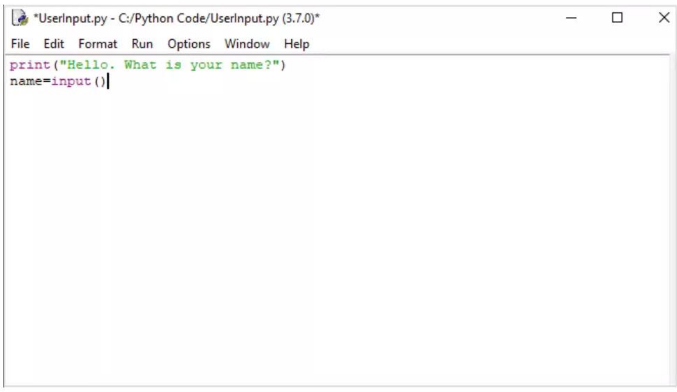

UserInput.py - C:/Python Code/UserInput.py (3.7.0)
File Edit Format Run Options Window Help

```
print("Hello.")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
print("Welcome", firstname, surname+". I hope you're well today.")
```

STEP 5

You don’t always have to include quoted text within the input command. For example, you can ask the user their name, and have the input in the line below:

```
print(“Hello. What’s your name?”)
name=input()
```

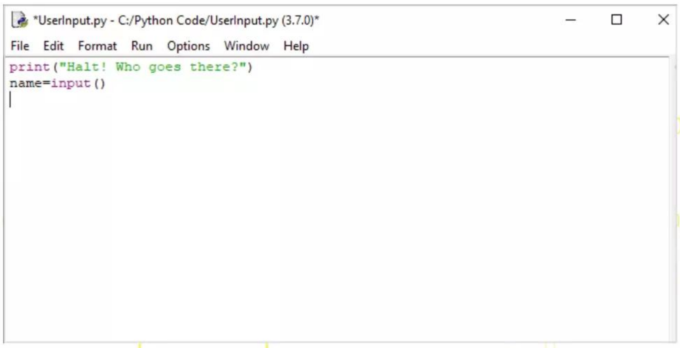

UserInput.py - C:/Python Code/UserInput.py (3.7.0)
File Edit Format Run Options Window Help

```
print("Hello. What is your name?")
name=input()
```

STEP 6

The code from the previous step is often regarded as being a little neater than having a lengthy amount of text in the input command, but it’s not a rule that’s set in stone, so do as you like in these situations. Expanding on the code, try this:

```
print(“Halt! Who goes there?”)
name=input()
```

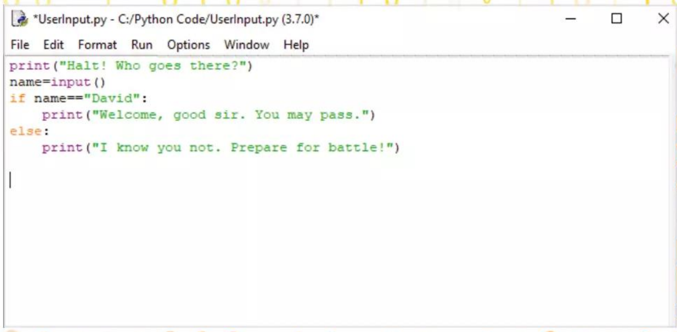

UserInput.py - C:/Python Code/UserInput.py (3.7.0)
File Edit Format Run Options Window Help

```
print("Halt! Who goes there?")
name=input()
```

STEP 7

It’s a good start to a text adventure game, perhaps? Now we can expand on it, and use the raw input from the user to flesh out the game a little:

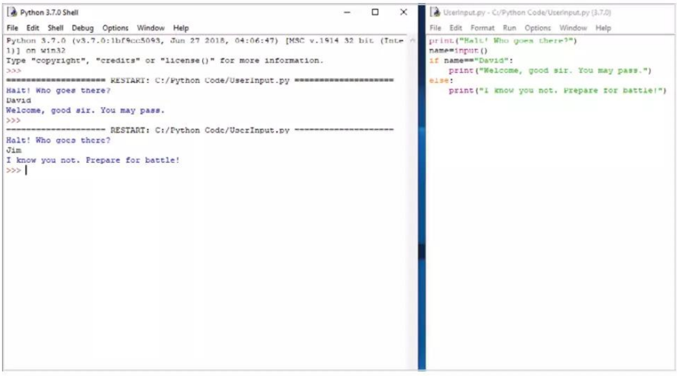
```
if name==”David”:
   print(“Welcome, good sir. You may pass.”)
else:
  print(“I know you not. Prepare for battle!”)
```


UserInput.py - C:/Python Code/UserInput.py (3.7.0)
File Edit Format Run Options Window Help

```
print("Halt! Who goes there?")
name=input()
if name=="David":
    print("Welcome, good sir. You may pass.")
else:
    print("I know you not. Prepare for battle!")
```

STEP 8

What you’ve created here is a condition, based on the user’s input. In short, we’re using the input from the user and measuring it against a condition. Therefore, if the user enters David as their name, the guard will allow them to pass unhindered. If, however, they enter a name other than David, the guard challenges them to a fight.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
UserInput.py - C:/Python Code/UserInput.py (3.7.0)
File Edit Format Run Options Window Help

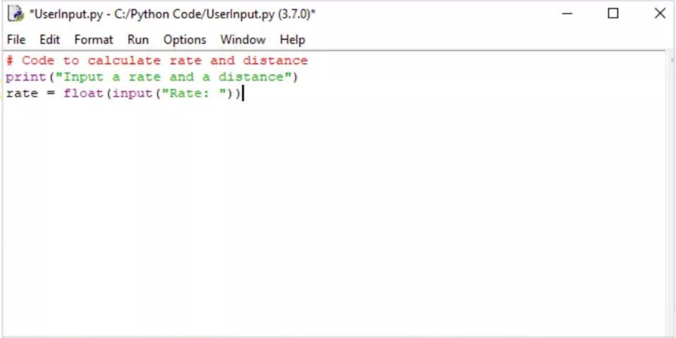
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Python Code/UserInput.py ====================
Halt! Who goes there?
David
Welcome, good sir. You may pass.
>>>
==================== RESTART: C:/Python Code/UserInput.py ====================
Halt! Who goes there?
Jim
I know you not. Prepare for battle!
>>> 
```

```
print("Halt! Who goes there?")
name=input()
if name=="David":
    print("Welcome, good sir. You may pass.")
else:
    print("I know you not. Prepare for battle!")
```

STEP 9

As you learned previously, any input from a user is automatically a string, so you’ll need to apply a TypeCast in order to turn it into something else. This creates some interesting additions to the input command. For example:

```
# Code to calculate rate and distance
print(“Input a rate and a distance”)
rate = float(input(“Rate: “))
```


UserInput.py - C:/Python Code/UserInput.py (3.7.0)
File Edit Format Run Options Window Help

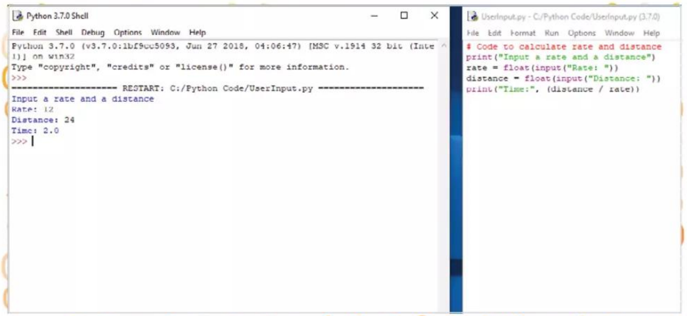
```
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
```

STEP 10

And to finalise the rate and distance code, we can add:

```
distance = float(input(“Distance: “))
print(“Time:”, (distance / rate))
```

Save and execute the code, and enter some numbers. Using the **float(input** element, we’ve told Python that anything entered is a floating point number rather than a string.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
UserInput.py - C:/Python Code/UserInput.py (3.7.0)
File Edit Format Run Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Python Code/UserInput.py ====================
Input a rate and a distance
Rate: 12
Distance: 24
Time: 2.0
>>> 
```

```
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
distance = float(input("Distance: "))
print("Time:", (distance / rate))
```

Creating Functions

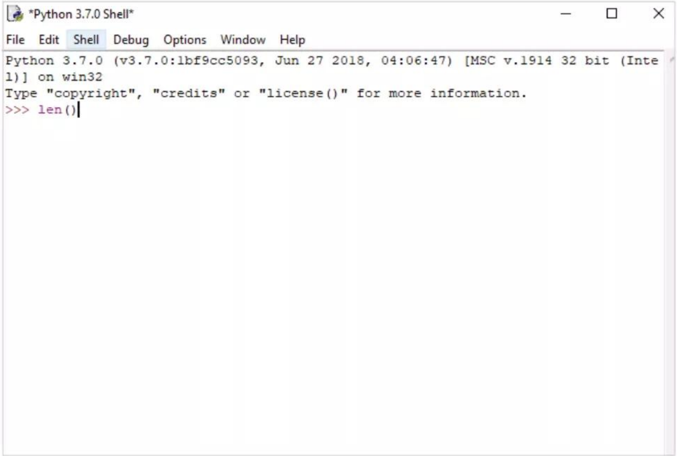
Now that you've mastered the use of variables and user input, the next step is to tackle functions. You've already used a few functions, such as the print command, but Python enables you to define your own function.

FUNKY FUNCTIONS

A function is a command that you enter into Python in order to do something. It's a little piece of self-contained code that takes data, works on it, and then returns the result.

STEP 1

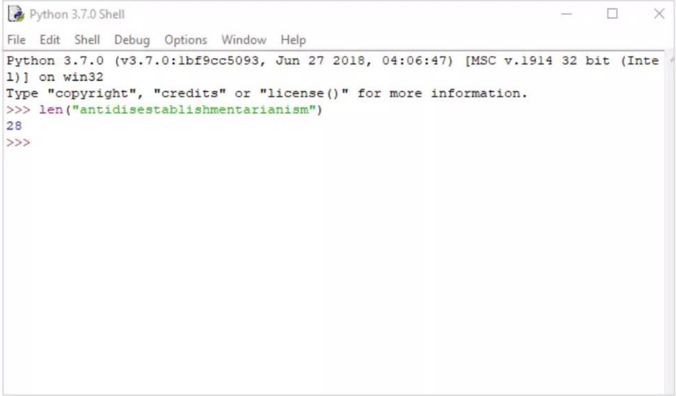
It's not only data that a function works on. Functions can do all manner of useful things in Python, such as sort data, change items from one format to another, and check the length or type of items. Basically, a function is a short word followed by brackets. For example, **len()**, **list()**, or **type()**.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> len()
```

STEP 2

A function takes data, usually a variable, works on it depending on what the function is programmed to do, and returns the end value. The data being worked on goes inside the brackets, so if you wanted to know how many letters are in the word antidisestablishmentarianism, then you'd enter: `len(“antidisestablishmentarianism”)`, and the number 28 would return.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

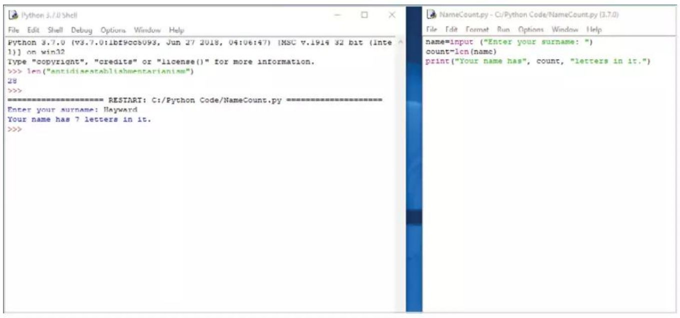
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>>
```

STEP 3

You can pass variables through functions in much the same manner. Let's assume you want the number of letters in a person's surname, you could use the following code (enter the text editor for this example):

```
name=input (“Enter your surname: ”)
count=len(name)
print (“Your surname has“, count, “letters in it.”)
```

Press **F5** and save the code to execute it.

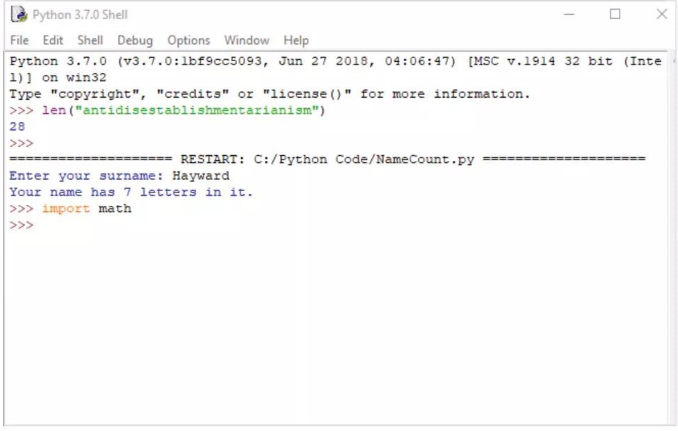

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
NameCount.py - C:/Python Code/NameCount.py (3.7.0)
File Edit Format Run Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>>
==================== RESTART: C:/Python Code/NameCount.py ====================
Enter your surname: Hayward
Your name has 7 letters in it.
>>>
```

```
name=input ("Enter your surname: ")
count=len(name)
print("Your name has", count, "letters in it.")
```

STEP 4

Python has tens of functions built into it, far too many to get into in the limited space available here. However, to view the list of built-in functions available to Python 3, navigate to **https://docs.python.org/3/library/functions.html**. These are the pre-defined functions, but since users have created many more, they're not the only ones available.

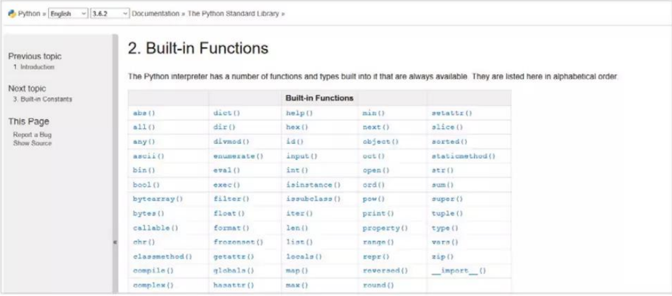

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>>
==================== RESTART: C:/Python Code/NameCount.py ====================
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>>
```

STEP 5 Additional functions can be added to Python through modules. Python has a vast range of modules available that can cover numerous programming duties. They add functions and can be imported as and when required. For example, to use advanced Mathematics functions enter:

```
import math
```

Once entered, you'll have access to all the Math module functions.


Python » English 3.6.2 Documentation » The Python Standard Library »
Previous topic
1. Introduction
Next topic
3. Built-in Constants
This Page
Report a Bug
Show Source

2. Built-in Functions

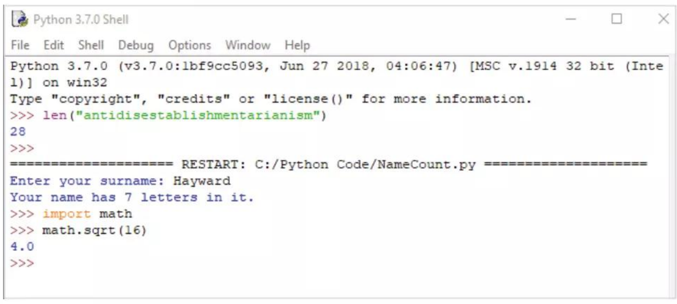
The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

STEP 6 To use a function from a module, enter the name of the module, followed by a full stop, then the name of the function. For instance, using the math module, since we've just imported it into Python, we can utilise the square root function. To do so, enter:

```
math.sqrt(16)
```

As you can see, the code is presented as **module.function(data)**.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>>
=================== RESTART: C:/Python Code/NameCount.py ====================
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>> math.sqrt(16)
4.0
>>>
```

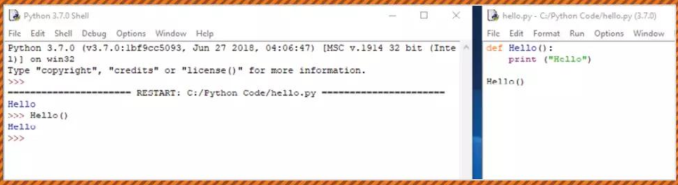
FORGING FUNCTIONS

There are many different functions, created by other Python programmers, which you can import and you'll undoubtedly come across some excellent examples in the future. However, you can also create your own with the def command.

STEP 1 Choose **File > New File** to enter the editor, let's create a function called Hello that will greet a user. Enter:

```
def Hello():
    print ("Hello")

Hello()
```

Press **F5** to save and run the script. You'll see Hello in the Shell, type in **Hello()** and it'll return the new function.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
hello.py - C:/Python Code/hello.py (3.7.0)
File Edit Format Run Options Window

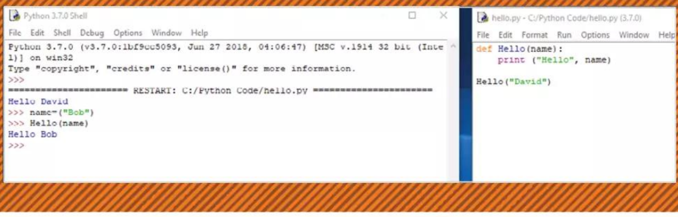
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:/Python Code/hello.py =====================
Hello
>>> Hello()
Hello
>>>
```

```
def Hello():
    print ("Hello")

Hello()
```

STEP 2 Let's now expand the function to accept a variable, the user's name for example. Edit your script to read:

```
def Hello(name):
    print ("Hello", name)

Hello("David")
```

This will now accept the variable name, otherwise it will print Hello David. In the Shell, enter: `name=("Bob")`, then, `Hello(name)`. Your function can now pass variables through it.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
hello.py - C:/Python Code/hello.py (3.7.0)
File Edit Format Run Options Window Help

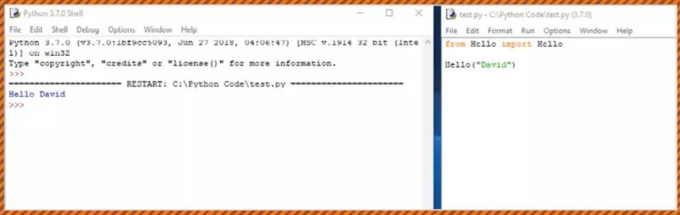
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Python Code/hello.py ======================
Hello David
>>> name=("Bob")
>>> Hello(name)
Hello Bob
>>>
```

```
def Hello(name):
    print ("Hello", name)

Hello("David")
```

STEP 3 To modify it further, delete the **Hello("David")** line, the last line in the script, and press Ctrl+S to save the new script. Close the Editor and create a new file (**File > New File**). Enter the following:

```
from Hello import Hello

Hello("David")
```

Press **F5** to save and execute the code.

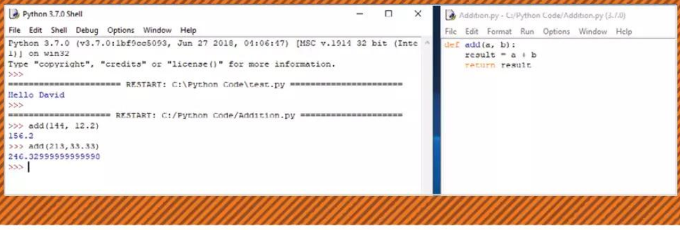

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
test.py - C:\Python Code\test.py (3.7.0)
File Edit Format Run Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:\Python Code\test.py =====================
Hello David
>>>
```

```
from Hello import Hello

Hello("David")
```

STEP 4 What you've just done is import the Hello function from the saved Hello.py program, and then used it to say hello to David. This is how modules and functions work, you import the module then use the function. Try this one, and modify it for extra credit:

```
def add(a, b):
    result = a + b
    return result
```


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Addition.py - C:/Python Code/Addition.py (3.7.0)
File Edit Format Run Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: C:\Python Code\test.py =====================
Hello David
>>>
=================== RESTART: C:/Python Code/Addition.py ===================
>>> add(144, 12.2)
156.2
>>> add(213,33.33)
246.32999999999998
>>>
```

```
def add(a, b):
    result = a + b
    return result
```

Conditions and Loops

Conditions and loops are what make a program interesting, they can be simple or rather complex. How you use them depends greatly on what the program is trying to achieve, they could be the number of lives left in a game, or just displaying a countdown.

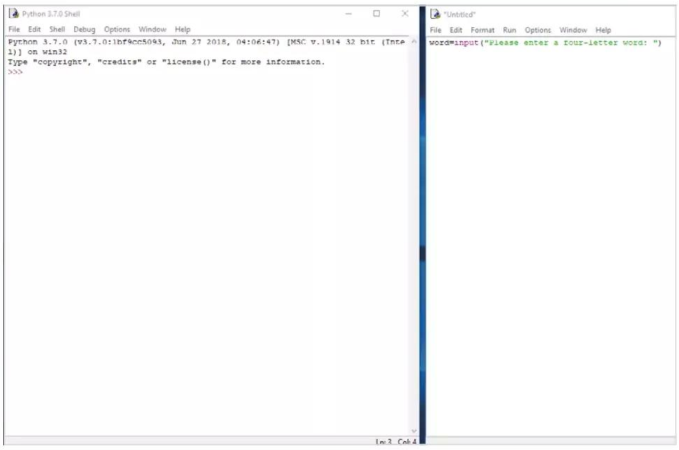
TRUE CONDITIONS

Keeping conditions simple, to begin with, makes learning to program a more enjoyable experience. Let's start then by checking if something is TRUE, then doing something else if it isn't.

STEP 1

Let's create a new Python program that will ask the user to input a word, then check it to see if it's a four-letter word or not. Start with **File > New File**, and begin with the input variable:

```
word=input(“Please enter a four-letter word: “)
```


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Untitled
File Edit Format Run Options Window Help
Ln: 3 Col: 4

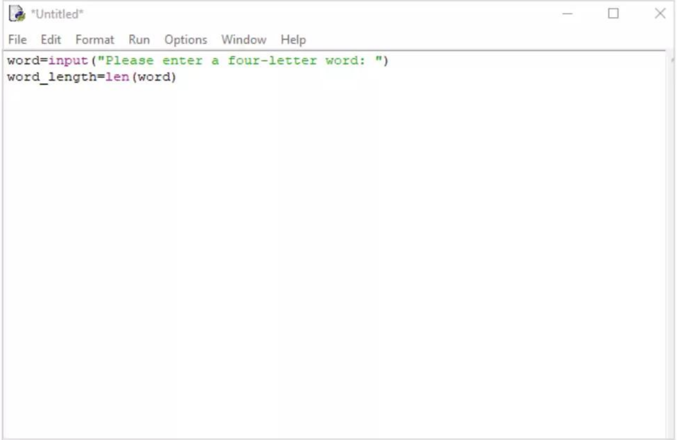
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
word=input("Please enter a four-letter word: ")
```

STEP 2

Now we can create a new variable, then use the len function and pass the word variable through it to get the total number of letters the user has just entered:

```
word=input(“Please enter a four-letter word: “)
word_length=len(word)
```


Untitled
File Edit Format Run Options Window Help

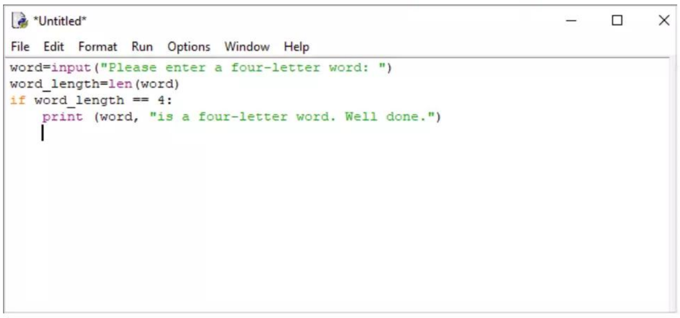
```
word=input("Please enter a four-letter word: ")
word_length=len(word)
```

STEP 3

Now we'll use an if statement to check if the word_length variable is equal to four, and print a friendly conformation if it applies to the rule:

```
word=input(“Please enter a four-letter word: “)
word_length=len(word)
if word_length == 4:
   print (word, “is a four-letter word. Well done.”)
```

The double equal sign (==) check if something is equal to something else.

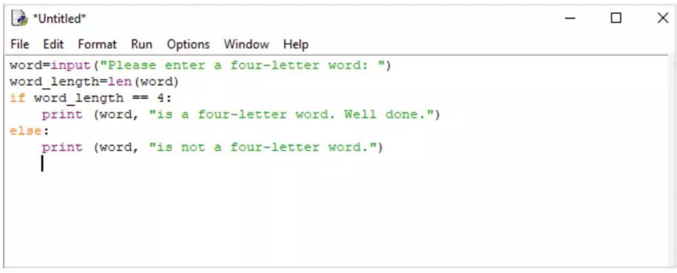

Untitled
File Edit Format Run Options Window Help

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
```

STEP 4

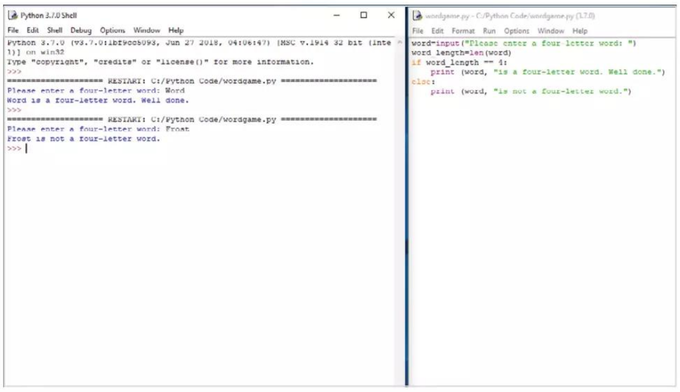
The colon at the end of if tells Python that if this statement is true, do everything after the colon that's indented. Next, move the cursor back to the beginning of the Editor:

```
word=input(“Please enter a four-letter word: “)
word_length=len(word)
if word_length == 4:
  print (word, “is a four-letter word. Well 
done.”)
else:
   print (word, “is not a four-letter word.”)
```


Untitled
File Edit Format Run Options Window Help

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
else:
    print (word, "is not a four-letter word.")
```

STEP 5 Press **F5** and save the code to execute it. Enter a four-letter word in the Shell to begin with, you should have the returned message that the word is four letters. Now press **F5** again, and re-run the program, but this time, enter a five-letter word. The Shell will display that it's not a four-letter word.

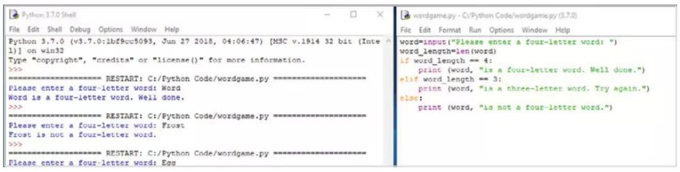

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
wordgame.py - C:/Python Code/wordgame.py (3.7.0)
File Edit Format Run Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Python Code/wordgame.py ====================
Please enter a four-letter word: Word
Word is a four-letter word. Well done.
>>>
==================== RESTART: C:/Python Code/wordgame.py ====================
Please enter a four-letter word: Frost
Frost is not a four-letter word.
>>> |
```

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
else:
    print (word, "is not a four-letter word.")
```

STEP 6 Now expand the code to include other conditions. Eventually, it could become quite complex. We've added a condition for three-letter words:

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
   print (word, "is a four-letter word. Well
done.")
elif word_length == 3:
   print (word, "is a three-letter word. Try again.")
else:
   print (word, "is not a four-letter word.")
```


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
wordgame.py - C:/Python Code/wordgame.py (3.7.0)
File Edit Format Run Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: C:/Python Code/wordgame.py ====================
Please enter a four-letter word: Word
Word is a four-letter word. Well done.
>>>
==================== RESTART: C:/Python Code/wordgame.py ====================
Please enter a four-letter word: Frost
Frost is not a four-letter word.
>>>
==================== RESTART: C:/Python Code/wordgame.py ====================
Please enter a four-letter word: Egg
```

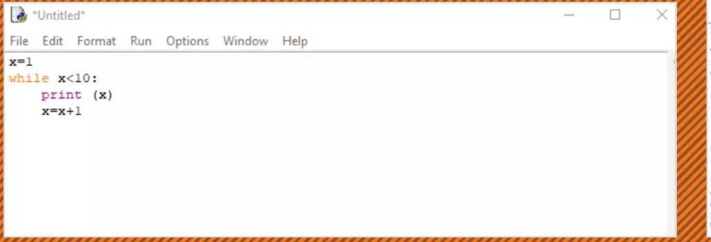
```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
elif word_length == 3:
    print (word, "is a three-letter word. Try again.")
else:
    print (word, "is not a four-letter word.")
```

LOOPS

Although a loop looks quite similar to a condition, they are somewhat different in their operation. A loop will run through the same block of code a number of times, usually with the support of a condition.

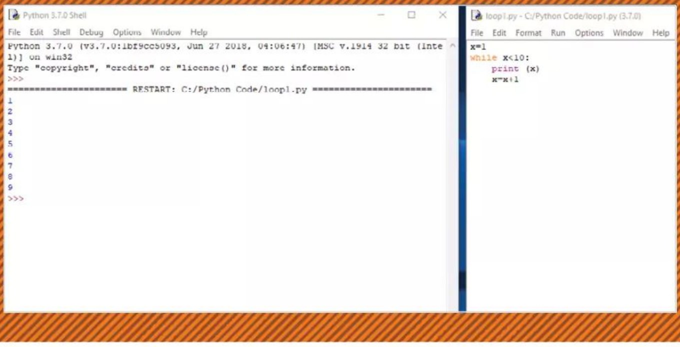
STEP 1 Let's start with a simple **while** statement. Like **if**, this will check to see if something is TRUE, then run the indented code:

```
x = 1
while x < 10:
  print (x)
  x = x + 1
```


Untitled
File Edit Format Run Options Window Help

```
x=1
while x<10:
    print (x)
    x=x+1
```

STEP 2 The difference between **if** and **while** is that when **while** gets to the end of the indented code, it goes back and checks the statement is still true. In our example x is less than 10. With each loop, it prints the current value of x, then adds one to that value. When x does eventually equal 10 it'll stop.

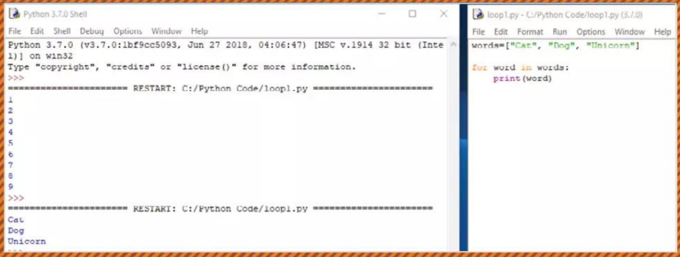

Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
loop1.py - C:/Python Code/loop1.py (3.7.0)
File Edit Format Run Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Python Code/loop1.py ======================
1
2
3
4
5
6
7
8
9
>>>
```

```
x=1
while x<10:
    print (x)
    x=x+1
```

STEP 3 The **for** loop, is another example. **For** is used to loop over a range of data, usually a list stored as variables inside square brackets. For example:

```
words=["Cat", "Dog", "Unicorn"]

for word in words:
   print (word)
```


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
loop1.py - C:/Python Code/loop1.py (3.7.0)
File Edit Format Run Options Window Help

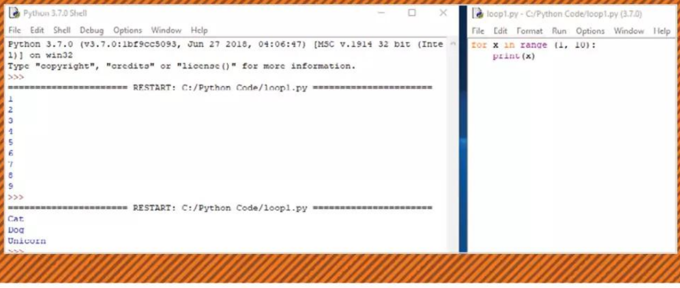
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Python Code/loop1.py ======================
1
2
3
4
5
6
7
8
9
>>>
====================== RESTART: C:/Python Code/loop1.py ======================
Cat
Dog
Unicorn
```

```
words=["Cat", "Dog", "Unicorn"]

for word in words:
    print(word)
```

STEP 4 The **for** loop can also be used in the countdown example by using the **range** function:

```
for x in range (1, 10):
  print (x)
```

The x=x+1 part isn't needed here, because the **range** function creates a list between the first and last numbers used.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
loop1.py - C:/Python Code/loop1.py (3.7.0)
File Edit Format Run Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: C:/Python Code/loop1.py ======================
1
2
3
4
5
6
7
8
9
>>>
====================== RESTART: C:/Python Code/loop1.py ======================
Cat
Dog
Unicorn
```

```
for x in range (1, 10):
    print(x)
```

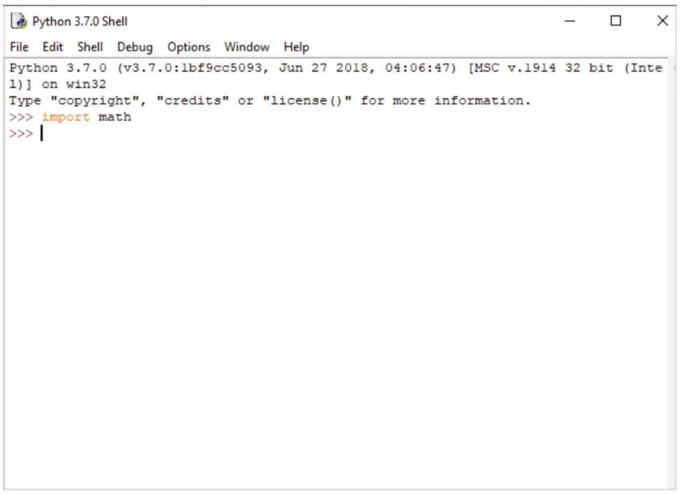
Python Modules

We've mentioned modules previously, using the Math module as an example, but since using modules is such a large part of getting the most from Python it's worth dedicating a little more time to them.

MASTERING MODULES

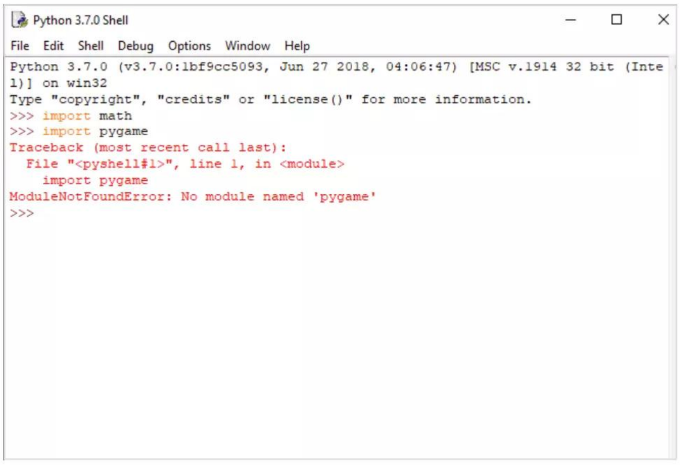
Think of modules as an extension that's imported into your Python code to enhance and extend its capabilities. There are countless modules available, and as we've seen, you can even make your own.

STEP 1 Although good, the built-in functions within Python are limited. The use of modules, however, allows us to make more sophisticated programs. As you are aware, modules are Python scripts that are imported, such as import math.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>> |
```

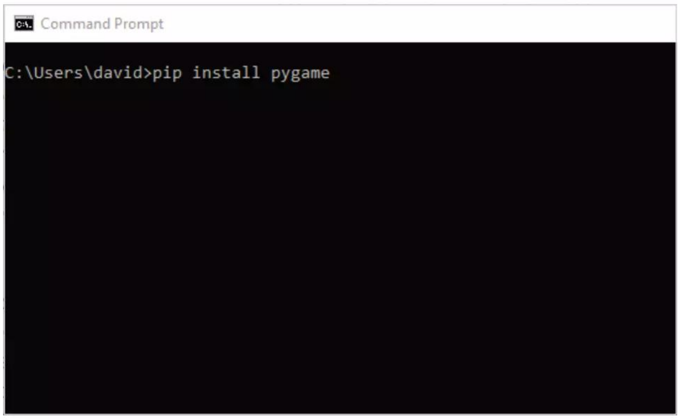
STEP 2 Some modules, especially on the Raspberry Pi, are included by default; the Math module is a prime example. Sadly, other modules aren't always available. A good example on non-Pi platforms is the Pygame module, which contains many functions to help create games. Try: **import pygame**.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>> import pygame
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    import pygame
ModuleNotFoundError: No module named 'pygame'
>>>
```

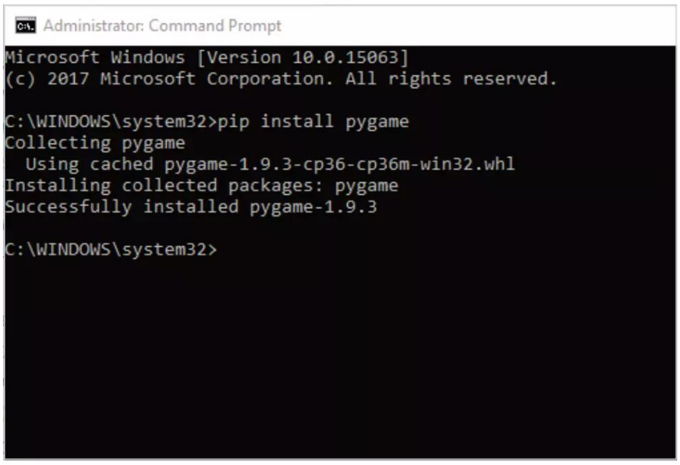
STEP 3 The result is an error in the IDLE Shell, as the Pygame module isn't recognised or installed in Python. To install a module we can use PIP (Pip Installs Packages). Close down the IDLE Shell and drop into a command prompt or Terminal session. At an elevated admin command prompt, enter:

```
pip install pygame
```



```
C:\Users\david>pip install pygame
```

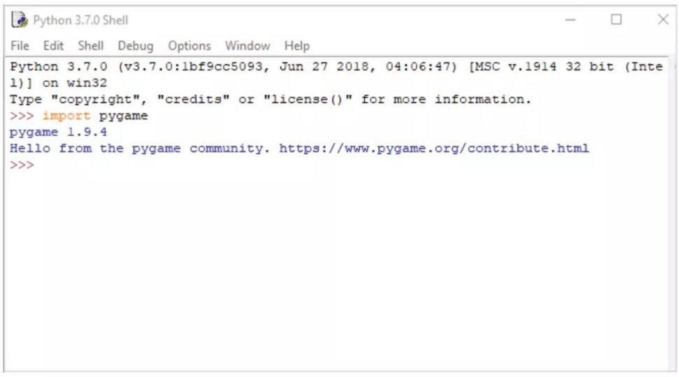
STEP 4 The PIP installation requires an elevated status due to it installing components at different locations. Start with a search for **CMD**, via the Start button, right-click the result, and then click **Run as Administrator**. Linux and Mac users can use the Sudo command, with **sudo pip install package**.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>pip install pygame
Collecting pygame
  Using cached pygame-1.9.3-cp36-cp36m-win32.whl
Installing collected packages: pygame
Successfully installed pygame-1.9.3

C:\WINDOWS\system32>
```

STEP 5

Close the command prompt or Terminal, and re-launch the IDLE Shell. When you now enter `import pygame`, the module will be imported into the code without any problems. You'll find that most code downloaded, or copied, from the Internet will contain a module, mainstream or unique, and their absence is commonly the source of errors in execution.


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import pygame
pygame 1.9.4
Hello from the pygame community. https://www.pygame.org/contribute.html
>>>
```

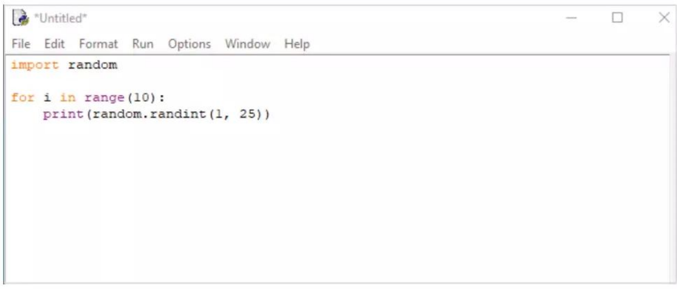
STEP 6

The modules contain the extra code needed to achieve a certain result within your own code, with which we've previously experimented. For example:

```
import random
```

Brings in the code from the Random number generator module. We can then use this module to create something like:

```
for i in range(10):
  print(random.randint(1, 25))
```

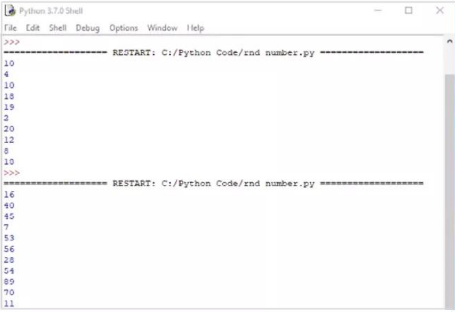

Untitled
File Edit Format Run Options Window Help

```
import random

for i in range(10):
    print(random.randint(1, 25))
```

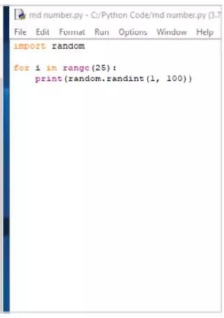
STEP 7

This code, when saved and executed, will display ten random numbers from 1 to 25. You can play around with the code to display more or less, and from a greater or lesser range. For example:

```
import random

for i in range(25):
  print(random.randint(1, 100))
```


Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

```
>>>
=================== RESTART: C:/Python Code/rnd number.py ===================
10
4
10
18
19
2
20
12
8
10
>>>
=================== RESTART: C:/Python Code/rnd number.py ===================
16
40
45
7
53
56
28
54
89
70
11
```

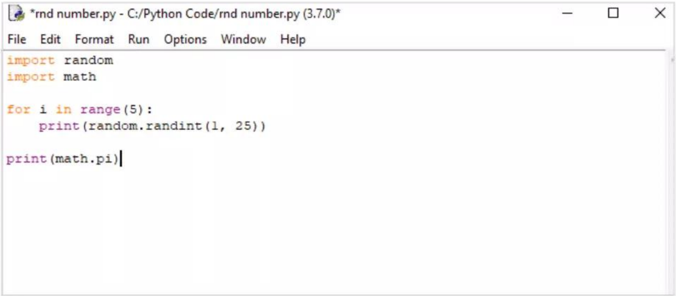

rnd number.py - C:/Python Code/rnd number.py (3.7
File Edit Format Run Options Window Help

```
import random

for i in range(25):
    print(random.randint(1, 100))
```

STEP 8

Multiple modules can be imported within your code. To extend our example, use:

```
import random
import math

for I in range(5):
  print(random.randint(1, 25))

print(math.pi)
```

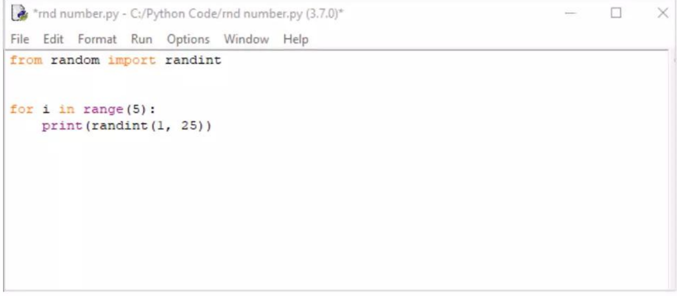

rnd number.py - C:/Python Code/rnd number.py (3.7.0)
File Edit Format Run Options Window Help

```
import random
import math

for i in range(5):
    print(random.randint(1, 25))

print(math.pi)
```

STEP 9

The result is a string of random numbers followed by the value of Pi, as pulled from the math module using the **print(math.pi)** function. You can also pull in certain functions from a module by using the **from** and **import** commands, such as:

```
from random import randint

for i in range(5):
  print(randint(1, 25))
```


rnd number.py - C:/Python Code/rnd number.py (3.7.0)
File Edit Format Run Options Window Help

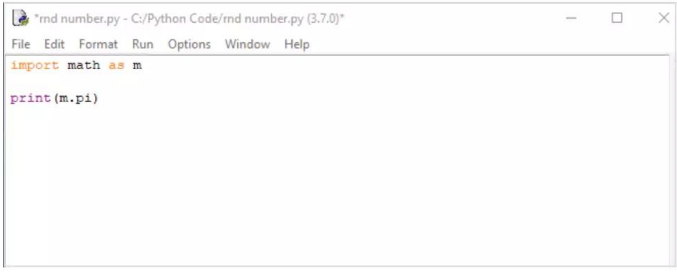
```
from random import randint


for i in range(5):
    print(randint(1, 25))
```

STEP 10

This helps create a more streamlined approach to programming. You can also use: **import module***, which will import everything defined within the named module. However, it's often regarded as a waste of resources, but it works nonetheless. Finally, modules can be imported as aliases:

```
import math as m

print(m.pi)
```

Of course, adding comments helps to tell others what's going on.

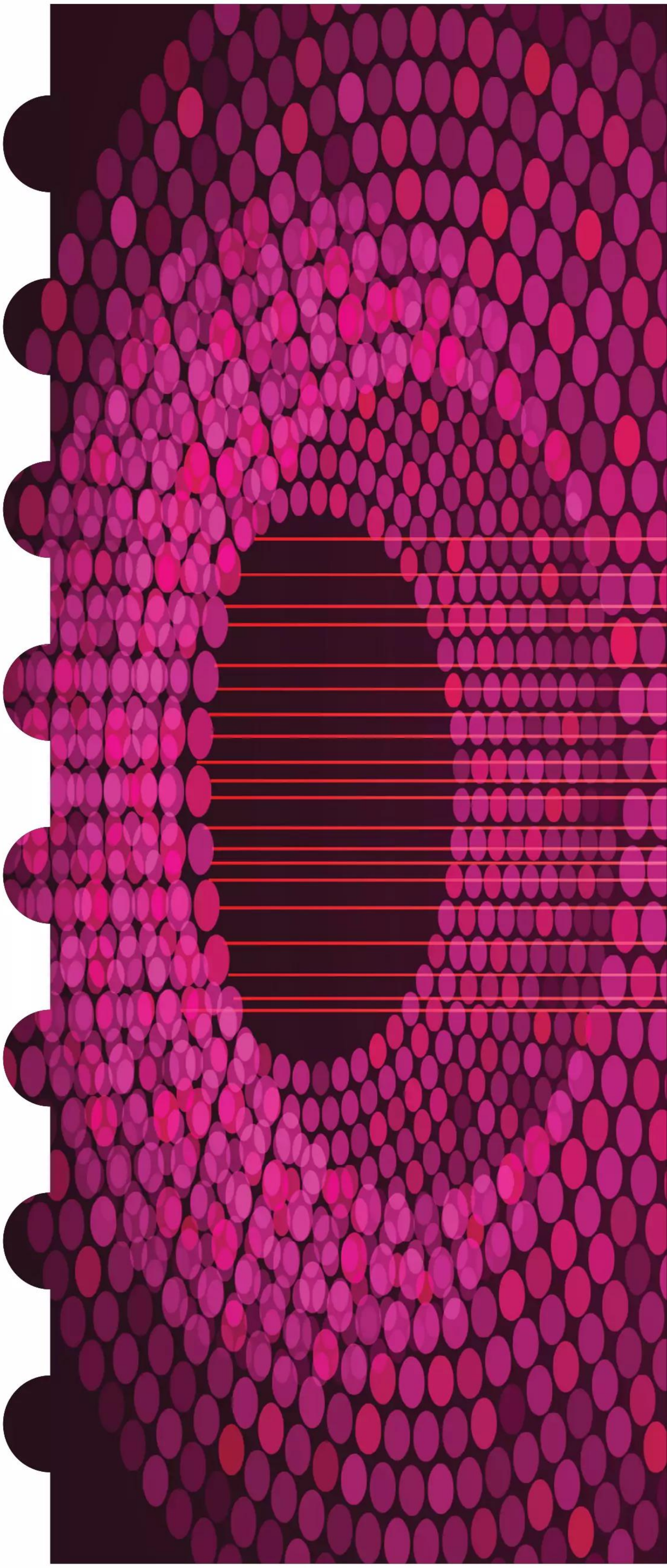

rnd number.py - C:/Python Code/rnd number.py (3.7.0)
File Edit Format Run Options Window Help

```
import math as m

print(m.pi)
```

Did you know that Windows has its own built-in scripting language? Batch files have been around since the early days of Windows, and while they are overshadowed by the might of Windows' modern graphical user interface, they are still there and still just as capable as they were thirty years ago.

Batch file programming is a skill that system administrators still use, so it's worth spending a bit of time learning how they work, and what you can do with them. This section introduces batch files, and covers user interactions, variables, loops and even a batch file quiz game to inject an element of fun.

96 What is a Batch File?
98 Getting Started with Batch Files
100 Getting an Output
102 Playing with Variables
104 Batch File Programming
106 Loops and Repetition
108 Creating a Batch File Game



Coding with Windows 10 Batch Files

What is a Batch File?

The Windows batch file has been around since the early days of DOS, and was once a critical element of actually being able to boot into a working system. There's a lot you can do with a batch file but let's just take a moment to see what one is.

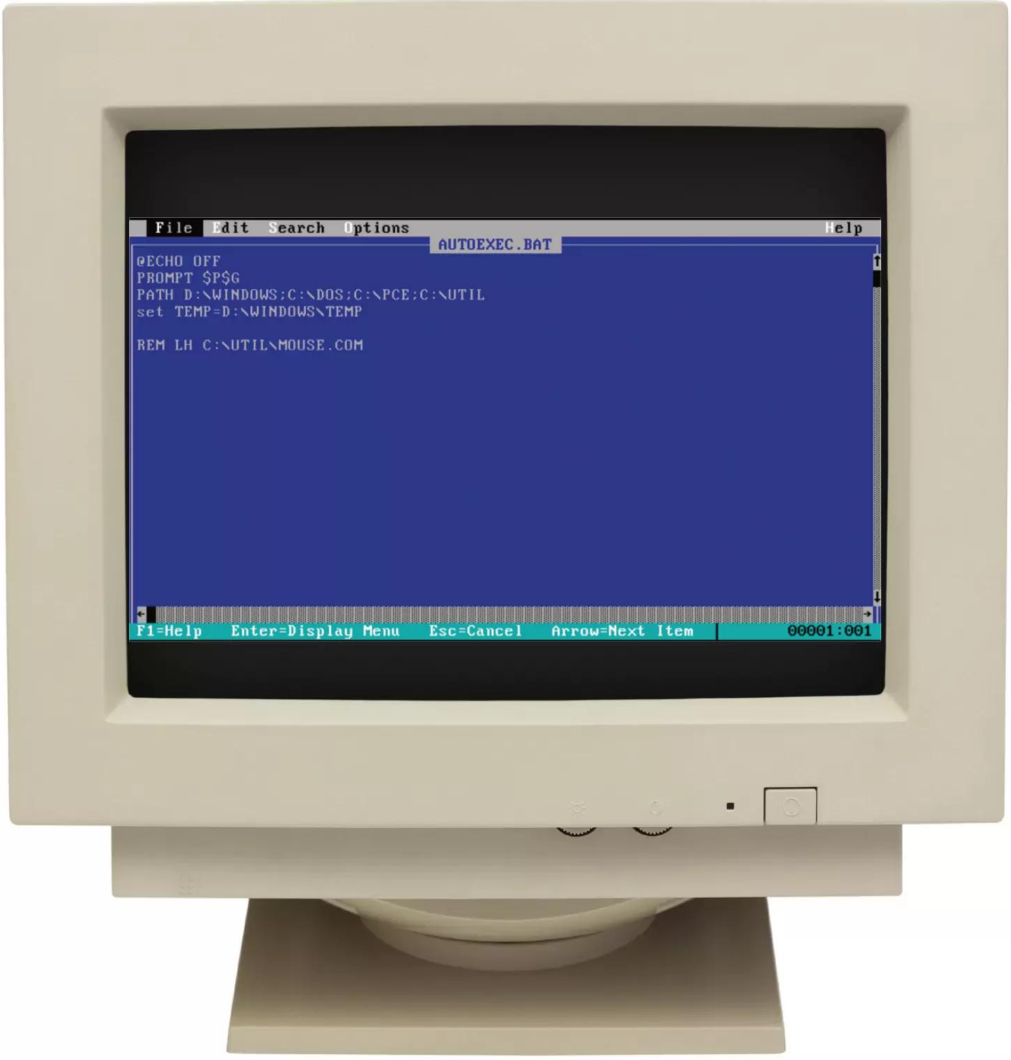
.BAT MAN

A Windows batch file is simply a script file that runs a series of commands, one line at a time, much in the same fashion as a Linux script. The series of commands are executed by the command line interpreter and stored in a plain text file with the .BAT extension; this signifies to Windows that it's an executable file, in this case, a script.

Batch files have been around since the earliest versions of Microsoft DOS. Although not exclusively a Microsoft scripting file, batch files are mainly associated with Microsoft's operating systems. In the early days, when a PC booted into a version of DOS (which produced a simple command prompt when powered up), the batch file was used in the form of a system file called Autoexec.bat. Autoexec.bat was a script that automatically executed (hence Autoexec) commands once the operating system had finished dealing with the Config.sys file.

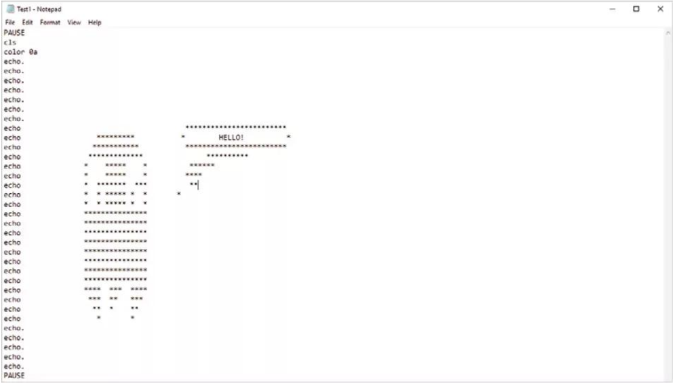
When a user powered up their DOS-based computer, and once the BIOS had finished checking the system memory and so on, DOS would look to the Config.sys file to load any specific display requirements and hardware drivers, allocate them a slot in the available memory, assign any memory managers and tell the system where the Command.com file, which is the command line interpreter for DOS, was. Once it had done that, then the Autoexec.bat file took over and ran through each line in turn, loading programs that would activate the mouse or optical drive into the memory areas assigned by the Config.sys file.

The DOS user of the day could opt to create different Autoexec.bat files depending on what they wanted to do. For example, if they wanted to play a game and have as much memory available as possible, they'd create a Config.sys and Autoexec.bat set of files that loaded the bare minimum of drivers and so on. If they needed access to the network, an Autoexec.bat file could be created to



```
@ECHO OFF
PROMPT $P$G
PATH D:\WINDOWS;C:\DOS;C:\PCE;C:\UTIL
set TEMP=D:\WINDOWS\TEMP

REM LH C:\UTIL\MOUSE.COM
```

The Autoexec.bat file was a PC user's first experience with a batch file.



```
PAUSE
cls
color 0a
echo.
echo.
echo.
echo.
echo.
echo.
echo.
echo                                      ************************
echo                  *********          *       HELLO!        *
echo                 ***********          ************************
echo                ************               **********
echo               *    *****    *          ******
echo               *    *****    *         ****
echo               *  *******  ***          **
echo               *  * ***** *  *      *
echo               *  * ***** *  *
echo               ***************
echo               ***************
echo               ***************
echo               ***************
echo               ***************
echo               ***************
echo               ***************
echo               ***************
echo               ****  ***  ****
echo                ***  **   ***
echo                 **  *    **
echo                  *       *
echo.
echo.
echo.
echo.
echo.
PAUSE
```

Batch files are plain text and often created using Notepad.

Diagnostics
Utilities
File Managers
Archivers
?

```
Install additional unpackers/extenders/utilities ?
[ highly recommended! ]

press Y for YES or N for NO [Y,N]?
```

Batch files were often used as utility programs, to help users with complex tasks.

load the network card driver and automatically gain access to the network. Each of these unique setups would be loaded on to a floppy disk and booted as and when required by the user.

The Autoexec.bat was the first such file many users came across in their PC-based computing lives; since many had come from a 16-bit or even 8-bit background; remember, this was the late eighties and early nineties. The batch file was the user's primary tool for automating tasks, creating shortcuts and adventure games and translating complex processes into something far simpler.

Nowadays however, a batch file isn't just for loading in drivers and such when the PC boots. You can use a batch file in the same way as any other scripting language file, in that you can program it to ask for user input and display the results on the screen; or save to a file and even send it to a locally or network attached printer. You can create scripts to back up your files to various locations, compare date stamps and only back up the most recently changed content as well as program the script to do all this automatically. Batch files are remarkably powerful and despite them not being as commonly used as they were during the older days of DOS, they are still there and can be utilised even in the latest version of Windows 10; and can be as complex or simple as you want them to be.

So what do you need to start batch file programming in Windows? Well, as long as you have Windows 10, or any older version of Windows for that matter, you can start batch file programming immediately. All you need is to be able to open Notepad and get to the command prompt of Windows. We show you how it all works, so read on.

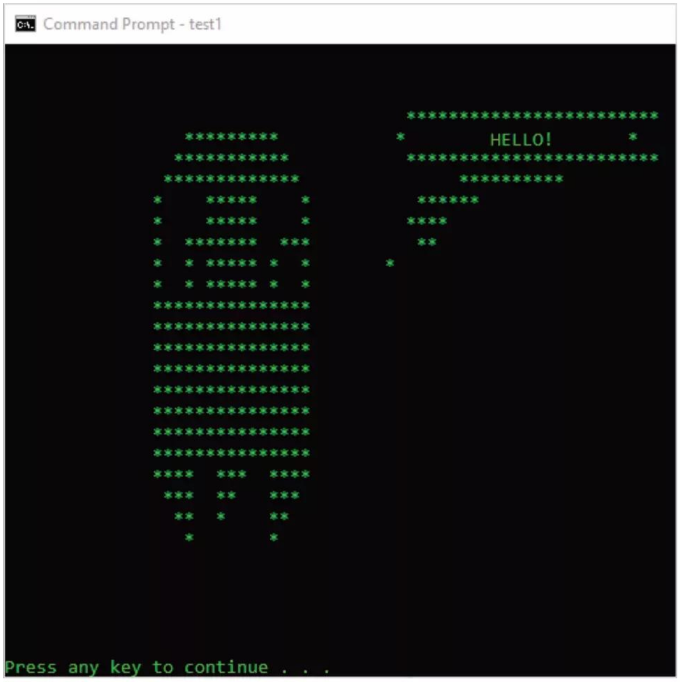
BATCH FILE POWER

Just like any other programming interface that can directly interrogate and manipulate the system, batch files require a certain amount of care when programming. It's hard to damage your system with a batch file, as the more important elements of the modern Windows system are protected by the User Account Control (UAC) security; UAC works by only allowing elevated privileges access to important system files. Therefore if you create a batch file that somehow deletes a system file, the UAC activates and stop the process.

However, if you're working in the command prompt with elevated privileges to begin with, as the Administrator, then the UAC won't question the batch file and continue regardless of what files are being deleted.

That said, you're not likely to create a batch file that intentionally wipes out your operating system. There are system controls in place to help prevent that; but it's worth mentioning as there are batch files available on the Internet that contain malicious code designed to create problems. Much like a virus, a rogue batch file (when executed with Administrator privileges) can cause much mayhem and system damage. In short, don't randomly execute any batch file downloaded from the Internet as an Administrator, without first reviewing what it does.

You can learn more about batch files in the coming pages, so don't worry too much about destroying your system with one. All this just demonstrates how powerful the humble batch file can be.



```
                                     ************************
                 *********          *       HELLO!        *
                ***********          ************************
               *************              **********
              *    *****    *          ******
              *    *****    *         ****
              *  *******  ***          **
              *  * ***** *  *      *
              *  * ***** *  *
              ***************
              ***************
              ***************
              ***************
              ***************
              ***************
              ***************
              ***************
              ****  ***  ****
               ***  **   ***
                **  *    **
                 *       *



Press any key to continue . . .
```

You can create complex batch files or simple ones that display ASCII images on screen.

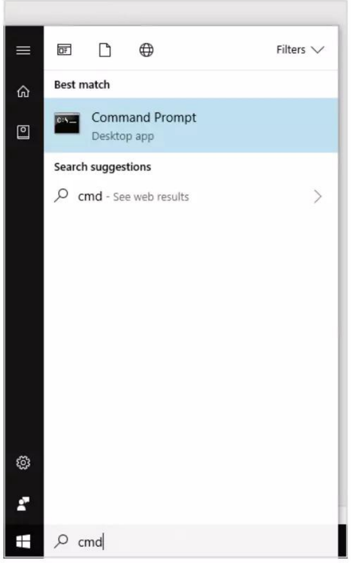
Getting Started with Batch Files

Before you begin to program with batch files, there are a few things you need to know. A batch file can only be executed once it has the .bat extension and editing one with Notepad isn't always straightforward.

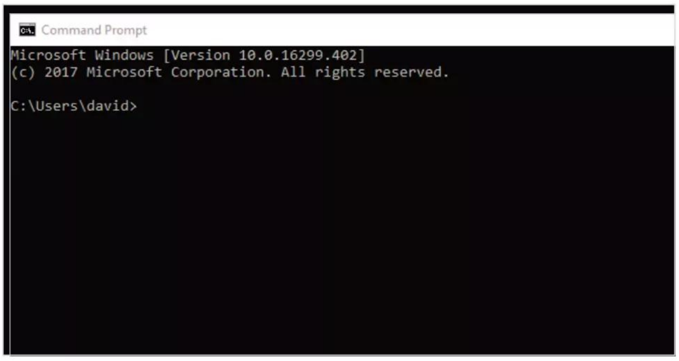
A NEW BATCH

Throughout this section on batch files we're going to be working with Notepad, the command prompt and within a folder called 'Batch Files'. To begin with, let's see how you get to the Windows command prompt.

STEP 1 The Windows command prompt may look a little daunting to the newcomer but it's simply another interface (or Shell) used to access the filesystem. You can go anywhere you like in the command prompt, as you would with the graphical interface. To begin, click on the Windows Start button and enter CMD into the search box.



STEP 2 Click on the search result labelled Command Prompt (Desktop App) and a new window pops up. The Command Prompt window isn't much to look at to begin with but you can see the Microsoft Windows version number and copyright information followed by the prompt itself. The prompt details the current directory or folder you're in, together with your username.

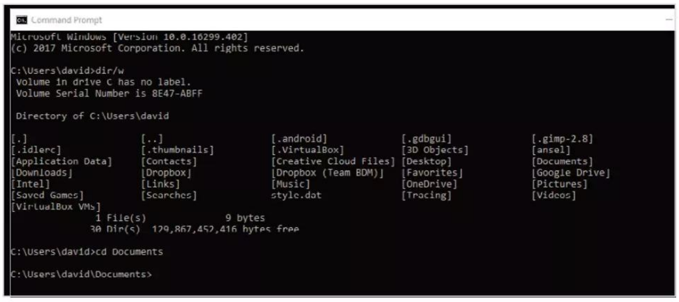


```
Microsoft Windows [Version 10.0.16299.402]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\david>
```

STEP 3 While at the command prompt window, enter: `dir/w.` This lists all the files and directories from where you are at the moment in the system. In this case, that's your Home directory that Windows assigns every user that logs in. You can navigate by using the CD command (Change Directory). Try:

```
cd Documents
```

Then press Return.



```
Microsoft Windows [Version 10.0.16299.402]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\david>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of C:\Users\david

[.]                    [..]                   [.android]             [.gdbgui]              [.gimp-2.8]
[.idlerc]              [.thumbnails]          [.VirtualBox]          [3D Objects]           [ansel]
[Application Data]     [Contacts]             [Creative Cloud Files] [Desktop]              [Documents]
[Downloads]            [Dropbox]              [Dropbox (Team BDM)]   [Favorites]            [Google Drive]
[Intel]                [Links]                [Music]                [OneDrive]             [Pictures]
[Saved Games]          [Searches]             style.dat              [Tracing]              [Videos]
[VirtualBox VMs]
               1 File(s)              9 bytes
              30 Dir(s)  129,867,452,416 bytes free

C:\Users\david>cd Documents

C:\Users\david\Documents>
```

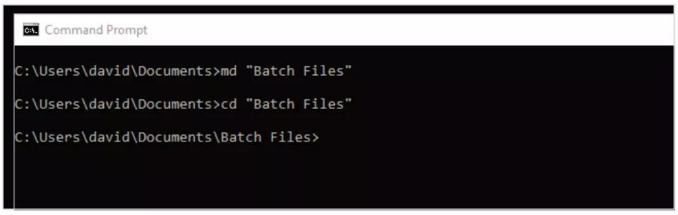
STEP 4 The prompt should change and display `\Documents>;` this means you're in the Documents directory. Now, create a new directory call Batch Files. Enter:

```
md "Batch Files"
```

You need the quotations because without them, Windows creates two directories: Batch and Files. Now change directory into the newly created Batch Files.

```
cd Batch Files
```

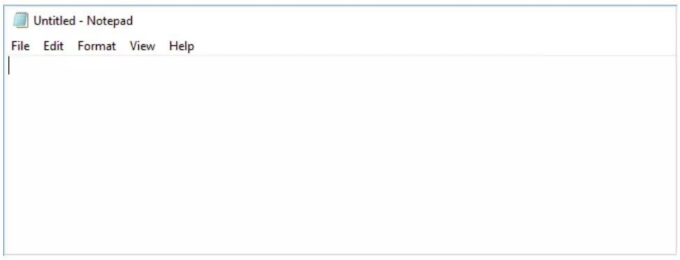
You won't need the quotes to change directories.



```
C:\Users\david\Documents>md "Batch Files"

C:\Users\david\Documents>cd "Batch Files"

C:\Users\david\Documents\Batch Files>
```

STEP 5

Now that you have the directory set up, where you store your batch files, here is how you can create one. Leave the command prompt window open and click on the Windows Start button again. This time enter Notepad and click on the search result to open the Notepad program. Notepad is a simple text editor but ideal for creating batch scripts with.

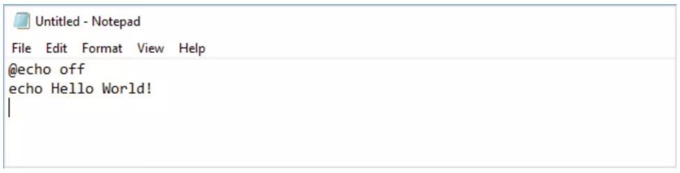

Untitled - Notepad
File Edit Format View Help

STEP 6

To create your first batch file, enter the following into Notepad:

```
@echo off
echo Hello World!
```

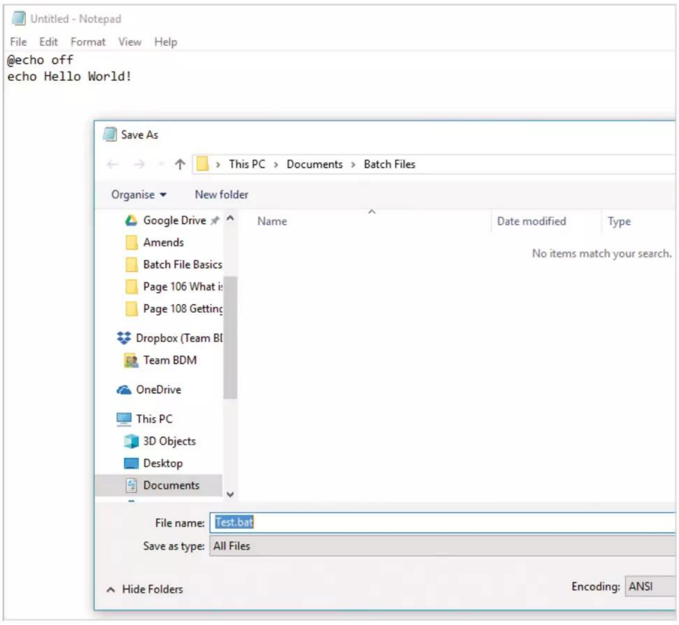
By default, a batch file displays all the commands that it runs through, line by line. What the @echo off command does is turn that feature off for the whole script; with the '@' (at) sign to apply that command to itself.


Untitled - Notepad
File Edit Format View Help

```
@echo off
echo Hello World!
```

STEP 7

When saving anything in Notepad the default extension is .txt, to denote a text file. However, you want the extension to be .bat. Click on File > Save As and navigate to the newly created Batch Files directory in Documents. Click the drop-down menu Save as Type, and select All Files from the menu. In File Name, call the file **Test.bat**.

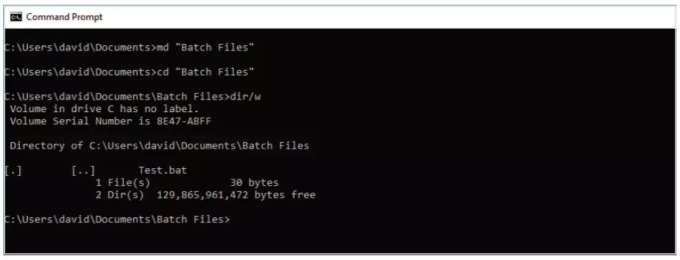

Untitled - Notepad
File Edit Format View Help
Save As
This PC › Documents › Batch Files
Organise ▾
New folder
Google Drive
Amends
Batch File Basics
Page 106 What i
Page 108 Gettin
Dropbox (Team B
Team BDM
OneDrive
This PC
3D Objects
Desktop
Documents
Name
Date modified
Type
No items match your search.
Hide Folders

```
@echo off
echo Hello World!
```

File name: Test.bat
Save as type: All Files
Encoding: ANSI

STEP 8

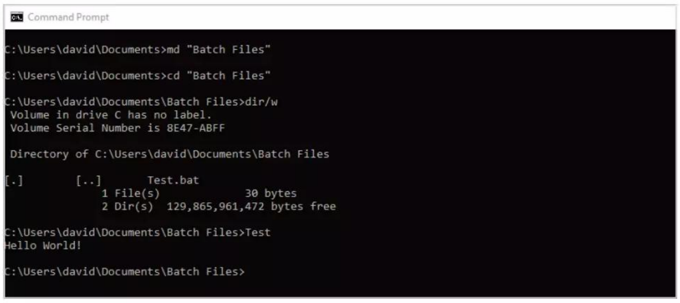
Back at the command prompt window, enter: `dir/w` again to list the newly created Test.bat file. By the way, the /w part of dir/w means the files are listed across the screen as opposed to straight down. Enter `dir` if you want (although you need more files to view) but it's considered easier to read with the /w flag.


Command Prompt

```
C:\Users\david\Documents>md "Batch Files"

C:\Users\david\Documents>cd "Batch Files"

C:\Users\david\Documents\Batch Files>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of C:\Users\david\Documents\Batch Files

[.]          [..]         Test.bat
               1 File(s)             30 bytes
               2 Dir(s)  129,865,961,472 bytes free

C:\Users\david\Documents\Batch Files>
```

STEP 9

To execute the batch file you've just created, simply enter its name, Test, in the command prompt window. You don't need to add the .bat part, as Windows recognises it as an executable file, and the only one with that particular name in the current directory. Press return and see how you're greeted with Hello World! in the command prompt.


Command Prompt

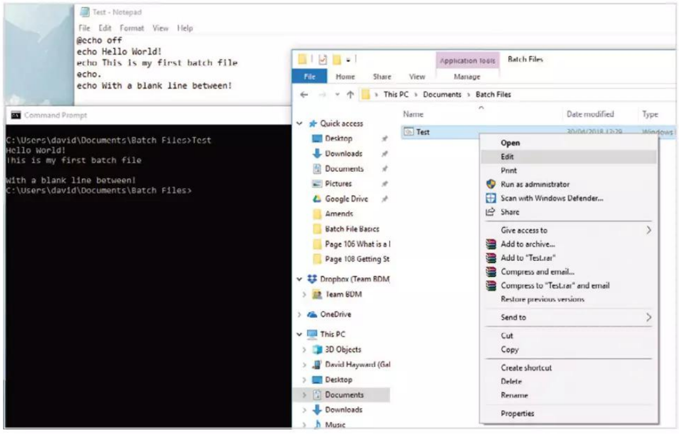
```
C:\Users\david\Documents>md "Batch Files"

C:\Users\david\Documents>cd "Batch Files"

C:\Users\david\Documents\Batch Files>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of C:\Users\david\Documents\Batch Files

[.]          [..]         Test.bat
               1 File(s)             30 bytes
               2 Dir(s)  129,865,961,472 bytes free

C:\Users\david\Documents\Batch Files>Test
Hello World!

C:\Users\david\Documents\Batch Files>
```

STEP 10

The echo command displays whatever is after it to the screen. Right-click the Test.bat file from Windows Explorer and select Edit to add more echo commands if you like. Try this:

```
@echo off
echo Hello World!
echo This is my first batch file
echo.
echo With a blank line between!
```

Remember to save each new change to the batch file.


Test - Notepad
File Edit Format View Help
Command Prompt
File Home Share View
Application Tools
Manage
Batch Files
This PC › Documents › Batch Files
Name
Date modified
Type
Quick access
Desktop
Downloads
Documents
Pictures
Google Drive
Amends
Batch File Basics
Page 106 What is a l
Page 108 Getting St
Dropbox (Team BDM
Team BDM
OneDrive
This PC
3D Objects
David Hayward (Gal
Desktop
Documents
Downloads
Music
Test
Open
Edit
Print
Run as administrator
Scan with Windows Defender...
Share
Give access to
Add to archive...
Add to "Test.rar"
Compress and email...
Compress to "Test.rar" and email
Restore previous versions
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties

```
@echo off
echo Hello World!
echo This is my first batch file
echo.
echo With a blank line between!
```

```
C:\Users\david\Documents\Batch Files>Test
Hello World!
This is my first batch file

With a blank line between!
C:\Users\david\Documents\Batch Files>
```

Getting an Output

While it's great having the command prompt window display what you're putting after the echo command in the batch file, it's not very useful at the moment, or interactive for that matter. Let's change up a gear and get some output.

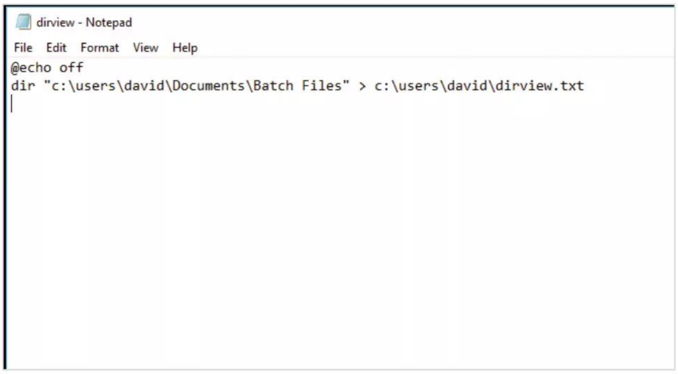
INPUT, OUTPUT

Batch files are capable of taking a normal Windows command and executing it, while also adding extra options and flags in to the equation.

STEP 1 Let's keep things simple to begin with. Create a new batch file called 'dirview.bat', short for Directory View. Start with the @echo off command and under that add:

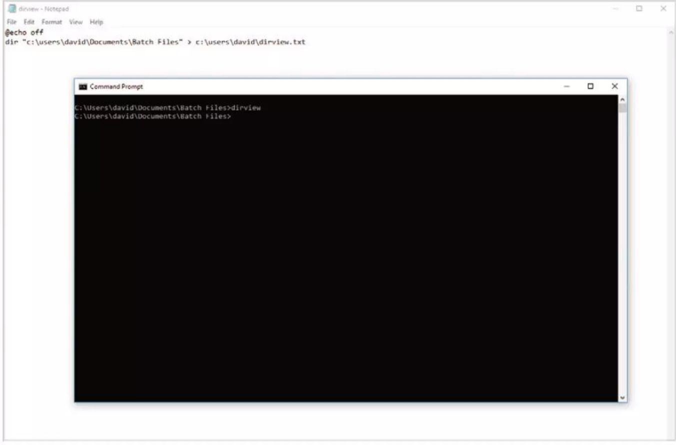
```
dir "c:\users\YOURNAME\Documents\Batch Files" >
c:\users\YOURNAME\dirview.txt
```

Substitute YOURNAME with your Windows username.



```
@echo off
dir "c:\users\david\Documents\Batch Files" > c:\users\david\dirview.txt
```

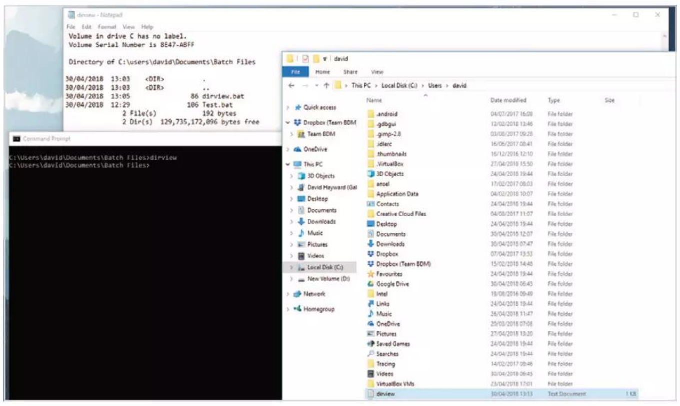
STEP 2 The new line uses the dir command to list the contents of the directory Batch Files, in your Home directory, dumping the output to a text file called `dirview.txt` in the root of your Home directory. This is done, so that the Windows UAC doesn't require elevated permissions, as everything is in your own Home area. Save and run the batch file.



```
@echo off
dir "c:\users\david\Documents\Batch Files" > c:\users\david\dirview.txt
```

```
C:\Users\david\Documents\Batch Files>dirview
C:\Users\david\Documents\Batch Files>
```

STEP 3 You have no doubt noticed that there is no indication that the batch file worked as there's no meaningful output on the screen. However, if you now open Explorer and browse to `c:\Users\YOURNAME`, remembering to substitute `YOURNAME` with your Windows username, and double-click the `dirview.txt` file, you can see the batch file's output.



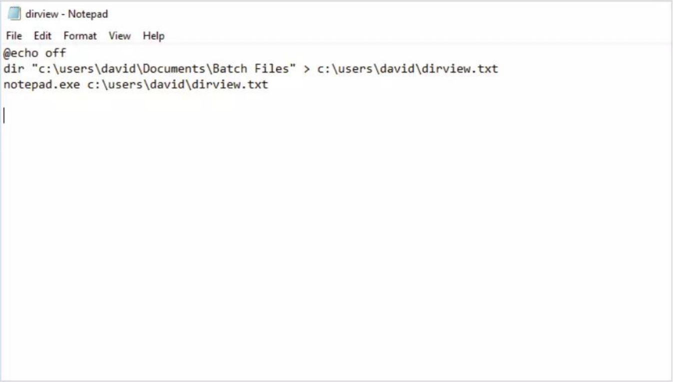
```
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of C:\users\david\Documents\Batch Files

30/04/2018  13:03    <DIR>          .
30/04/2018  13:03    <DIR>          ..
30/04/2018  13:05                86 dirview.bat
30/04/2018  12:29               106 Test.bat
               2 File(s)            192 bytes
               2 Dir(s)  129,735,172,096 bytes free
```

```
C:\Users\david\Documents\Batch Files>dirview
C:\Users\david\Documents\Batch Files>
```

STEP 4 If you want to automate the task of opening the text file that contains the output, add the following line to the batch file:

```
notepad.exe c:\users\YOURNAME\dirview.txt
```

Save the file and once again run from the command prompt. This time, it creates the output and automatically opens Notepad with the output contents.



```
@echo off
dir "c:\users\david\Documents\Batch Files" > c:\users\david\dirview.txt
notepad.exe c:\users\david\dirview.txt
```

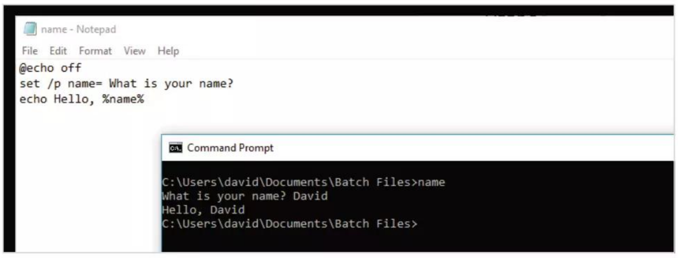
OUTPUT WITH VARIABLES

Variables offer a more interesting way of outputting something to the screen and create a higher level of interaction between the user and the batch file. Try this example below.

STEP 1 Create a new batch file and call it **name.bat**. Start with the `@echo off` command, then add the following lines:

```
set /p name= What is your name?
echo Hello, %name%
```

Note: there's a space after the question mark. This is to make it look neater on the screen. Save it and run the batch file.



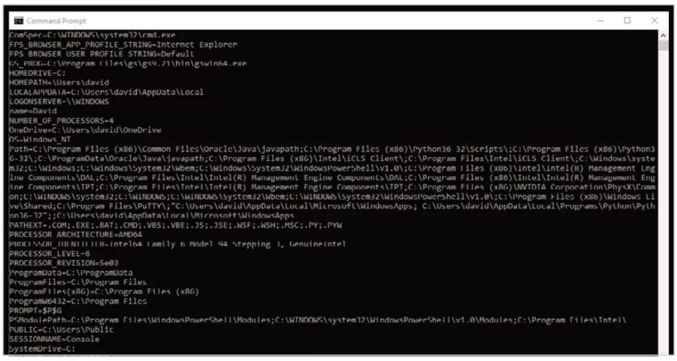
```
@echo off
set /p name= What is your name?
echo Hello, %name%
```

```
C:\Users\david\Documents\Batch Files>name
What is your name? David
Hello, David
C:\Users\david\Documents\Batch Files>
```

STEP 2 The set **/p** name creates a variable called name, with the **/p** part indicating that an '**=prompt string**' is to follow. The Set command displays, sets or removes system and environmental variables. For example, while in the command prompt window enter:

```
set
```

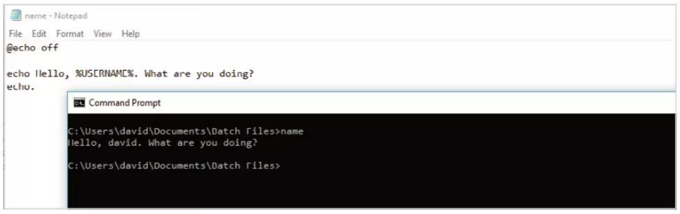
To view the current system variables. Note the name= variable we just created.



```
ComSpec=C:\WINDOWS\system32\cmd.exe
FPS_BROWSER_APP_PROFILE_STRING=Internet Explorer
FPS_BROWSER_USER_PROFILE_STRING=Default
GS_PROG=C:\Program Files\gs\gs9.21\bin\gswin64.exe
HOMEDRIVE=C:
HOMEPATH=\Users\david
LOCALAPPDATA=C:\Users\david\AppData\Local
LOGONSERVER=\\WINDOWS
name=David
NUMBER_OF_PROCESSORS=4
OneDrive=C:\Users\david\OneDrive
OS=Windows_NT
Path=C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Program Files (x86)\Python36 32\Scripts\;C:\Program Files (x86)\Python3
6-32\;C:\ProgramData\Oracle\Java\javapath;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\Windows\syste
m32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Intel\Intel(R) Management Eng
ine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Eng
ine Components\IPT;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Comm
on;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Windows Li
ve\Shared;C:\Program Files\PuTTY\;"C:\Users\david\AppData\Local\Microsoft\WindowsApps; C:\Users\david\AppData\Local\Programs\Python\Pyth
on36-32";;C:\Users\david\AppData\Local\Microsoft\WindowsApps
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW
PROCESSOR_ARCHITECTURE=AMD64
PROCESSOR_IDENTIFIER=Intel64 Family 6 Model 94 Stepping 3, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=5e03
ProgramData=C:\ProgramData
ProgramFiles=C:\Program Files
ProgramFiles(x86)=C:\Program Files (x86)
ProgramW6432=C:\Program Files
PROMPT=$P$G
PSModulePath=C:\Program Files\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules;C:\Program Files\Intel\
PUBLIC=C:\Users\Public
SESSIONNAME=Console
SystemDrive=C:
```

STEP 3 Variables stored with Set can be called with the **%VARIABLENAME% syntax**. In the batch file, we used the newly created %name% syntax to call upon the contents of the variable called name. Your username, for example, is stored as a variable. Try this in a batch file:

```
echo Hello, %USERNAME%. What are you doing?
```

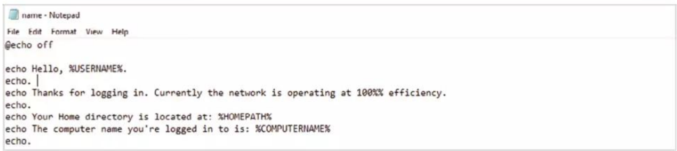


```
@echo off

echo Hello, %USERNAME%. What are you doing?
echo.
```

```
C:\Users\david\Documents\Batch Files>name
Hello, david. What are you doing?

C:\Users\david\Documents\Batch Files>
```

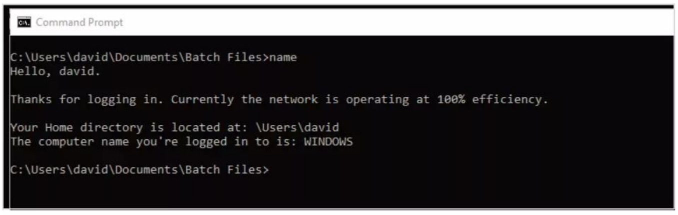
STEP 4 This is extremely useful if you want to create a unique, personal batch file that automatically runs when a user logs into Windows. Using the default systems variables that Windows itself creates, you can make a batch file that greets each user:

```
@echo off

echo Hello, %USERNAME%.
echo.
echo Thanks for logging in. Currently the network
is operating at 100%% efficiency.
echo.
echo Your Home directory is located at: %HOMEPATH%
echo The computer name you're logged in to is:
%COMPUTERNAME%
echo.
```



```
@echo off

echo Hello, %USERNAME%.
echo.
echo Thanks for logging in. Currently the network is operating at 100%% efficiency.
echo.
echo Your Home directory is located at: %HOMEPATH%
echo The computer name you're logged in to is: %COMPUTERNAME%
echo.
```

STEP 5 Save and execute the batch file changes; you can overwrite and still use name.bat if you want. The batch file takes the current system variables and reports them accordingly, depending on the user's login name and the name of the computer. Note: the double percent symbol means the percent sign will be displayed, and is not a variable.



```
C:\Users\david\Documents\Batch Files>name
Hello, david.

Thanks for logging in. Currently the network is operating at 100% efficiency.

Your Home directory is located at: \Users\david
The computer name you're logged in to is: WINDOWS

C:\Users\david\Documents\Batch Files>
```

STEP 6 Alternatively, you can run the batch file and display it on the user's desktop as a text file:

```
@echo off

echo Hello, %USERNAME%. > c:%HOMEPATH%\user.txt
echo. >> c:%HOMEPATH%\user.txt
echo Thanks for logging in. Currently the network
is operating at 100%% efficiency. >> c:%HOMEPATH%\
user.txt
echo. >> c:%HOMEPATH%\user.txt
echo Your Home directory is located at: %HOMEPATH%
>> c:%HOMEPATH%\user.txt
echo The computer name you're logged in to is:
%COMPUTERNAME% >> c:%HOMEPATH%\user.txt
echo. >> c:%HOMEPATH%\user.txt

notepad c:%HOMEPATH%\user.txt
```

The > outputs to a new file called user.txt, while the >> adds the lines within the file.

Playing with Variables

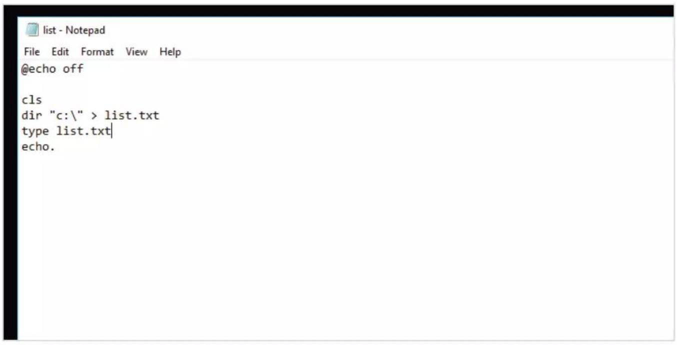
There’s a lot you can accomplish with both the system and environmental variables, alongside your own. Mixing the two can make for a powerful and extremely useful batch file and when combined with other commands, the effect is really impressive.

USING MORE VARIABLES

Here’s a good example of mixing system and environmental variables with some of your own creation, along with a number of external Windows commands.

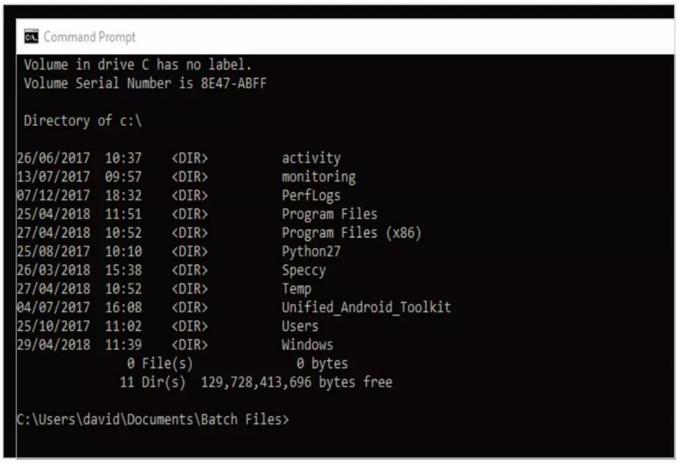
STEP 1 Create a new batch file called list.bat and start it off with the `@echo off` command. Begin by clearing the command prompt screen and displaying a list of the current directories on the computer:

```
cls
dir “c:\” > list.txt
type list.txt
echo.
```



```
@echo off

cls
dir "c:\" > list.txt
type list.txt
echo.
```

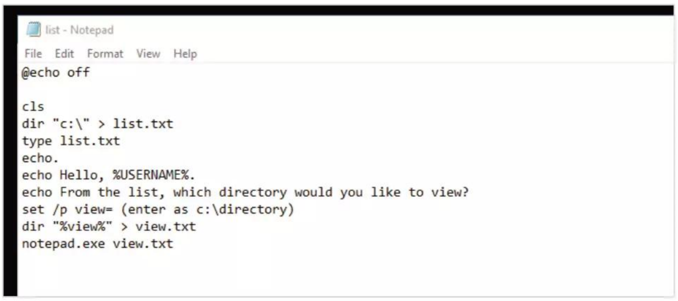
STEP 2 Save and execute the batch file. Within the command prompt you can see the contents of all the files and directories from the root of the C:\ drive; and as any user under Windows has permission to see this, there’s no UAC elevated privileges required.



```
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of c:\

26/06/2017  10:37    <DIR>          activity
13/07/2017  09:57    <DIR>          monitoring
07/12/2017  18:32    <DIR>          PerfLogs
25/04/2018  11:51    <DIR>          Program Files
27/04/2018  10:52    <DIR>          Program Files (x86)
25/08/2017  10:10    <DIR>          Python27
26/03/2018  15:38    <DIR>          Speccy
27/04/2018  10:52    <DIR>          Temp
04/07/2017  16:08    <DIR>          Unified_Android_Toolkit
25/10/2017  11:02    <DIR>          Users
29/04/2018  11:39    <DIR>          Windows
               0 File(s)              0 bytes
              11 Dir(s)  129,728,413,696 bytes free

C:\Users\david\Documents\Batch Files>
```

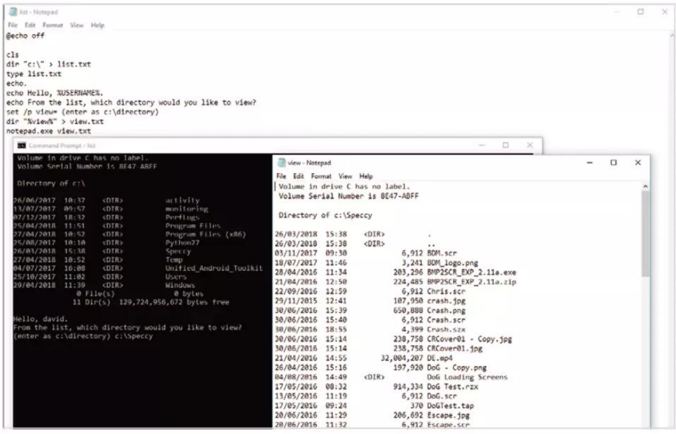
STEP 3 Now, create a batch file that displays the contents of any directory and post it as a text file to the user’s screen. Add the following to the list.bat batch file:

```
echo Hello, %USERNAME%.
echo From the list, which folder would you like to
view?
set /p view= (enter as c:\folder)
dir “%view%” > view.txt
notepad.exe view.txt
```



```
@echo off

cls
dir "c:\" > list.txt
type list.txt
echo.
echo Hello, %USERNAME%.
echo From the list, which directory would you like to view?
set /p view= (enter as c:\directory)
dir "%view%" > view.txt
notepad.exe view.txt
```

STEP 4 What’s happening here is the batch file asks the user to enter any of the directories displayed in the list it generated, in the form of ‘**c:\directory**’. Providing the user enters a valid directory, its contents are displayed as a text file. We created the view variable here along with %HOMEPATH%, to store the input and the text file.



```
@echo off

cls
dir "c:\" > list.txt
type list.txt
echo.
echo Hello, %USERNAME%.
echo From the list, which directory would you like to view?
set /p view= (enter as c:\directory)
dir "%view%" > view.txt
notepad.exe view.txt
```

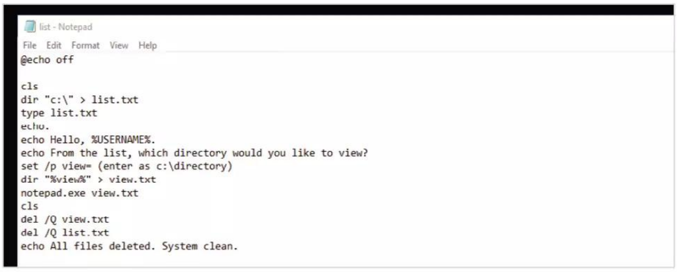
```
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of c:\

26/06/2017  10:37    <DIR>          activity
13/07/2017  09:57    <DIR>          monitoring
07/12/2017  18:32    <DIR>          PerfLogs
25/04/2018  11:51    <DIR>          Program Files
27/04/2018  10:52    <DIR>          Program Files (x86)
25/08/2017  10:10    <DIR>          Python27
26/03/2018  15:38    <DIR>          Speccy
27/04/2018  10:52    <DIR>          Temp
04/07/2017  16:08    <DIR>          Unified_Android_Toolkit
25/10/2017  11:02    <DIR>          Users
29/04/2018  11:39    <DIR>          Windows
               0 File(s)              0 bytes
              11 Dir(s)  129,724,956,672 bytes free

Hello, david.
From the list, which directory would you like to view?
(enter as c:\directory) c:\Speccy
```

```
 Volume in drive C has no label.
 Volume Serial Number is 8E47-ABFF

 Directory of c:\Speccy

26/03/2018  15:38    <DIR>          .
26/03/2018  15:38    <DIR>          ..
03/11/2017  09:30             6,912 BDM.scr
18/07/2017  11:46             3,241 BDM_logo.png
28/04/2016  11:34           203,296 BMP2SCR_EXP_2.11a.exe
21/04/2016  12:50           224,485 BMP2SCR_EXP_2.11a.zip
22/09/2016  12:59             6,912 Chris.scr
29/11/2015  12:41           107,950 crash.jpg
30/06/2016  15:39           650,888 Crash.png
30/06/2016  15:40             6,912 Crash.scr
30/06/2016  18:55             4,399 Crash.szx
30/06/2016  15:14           238,758 CRCover01 - Copy.jpg
30/06/2016  15:14           238,758 CRCover01.jpg
21/04/2016  14:55        32,004,207 DE.mp4
26/04/2016  15:16           197,920 DoG - Copy.png
04/08/2016  14:49    <DIR>          DoG Loading Screens
17/05/2016  08:32           914,334 DoG Test.rzx
13/05/2016  11:19             6,912 DoG.scr
17/05/2016  09:24               370 DoGTest.tap
20/06/2016  11:29           206,692 Escape.jpg
20/06/2016  11:32             6,912 Escape.scr
```

STEP 5

It's always a good idea, when creating text files for the user to temporarily view, to clean up after yourself. There's nothing worse than having countless, random text files cluttering up the file system. That being the case, let's clear up with:

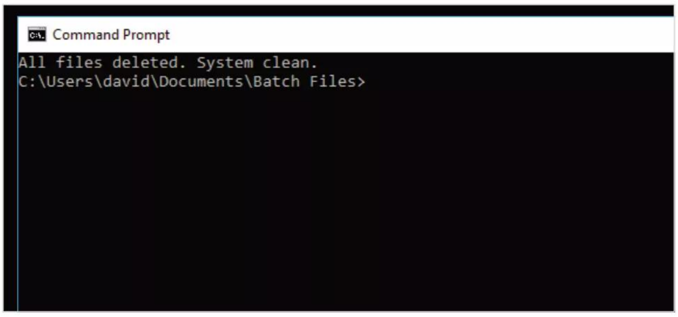
```
cls
del /Q view.txt
del /Q list.txt
echo All files deleted. System clean.
```


list - Notepad
File Edit Format View Help

```
@echo off

cls
dir "c:\" > list.txt
type list.txt
echo.
echo Hello, %USERNAME%.
echo From the list, which directory would you like to view?
set /p view= (enter as c:\directory)
dir "%view%" > view.txt
notepad.exe view.txt
cls
del /Q view.txt
del /Q list.txt
echo All files deleted. System clean.
```

STEP 6

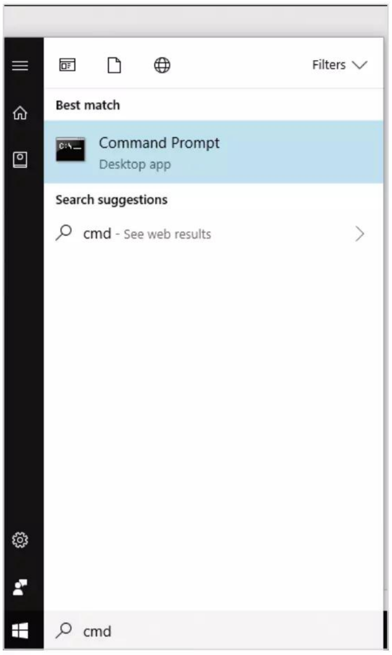
The additions to the batch file simply clear the command prompt window (using the cls command) and delete both the view.txt and list.txt files that were created by the batch file. The /Q flag in the del command means it deletes the files without any user input or notification. The final message informs the user that the files are removed.


Command Prompt

```
All files deleted. System clean.
C:\Users\david\Documents\Batch Files>
```

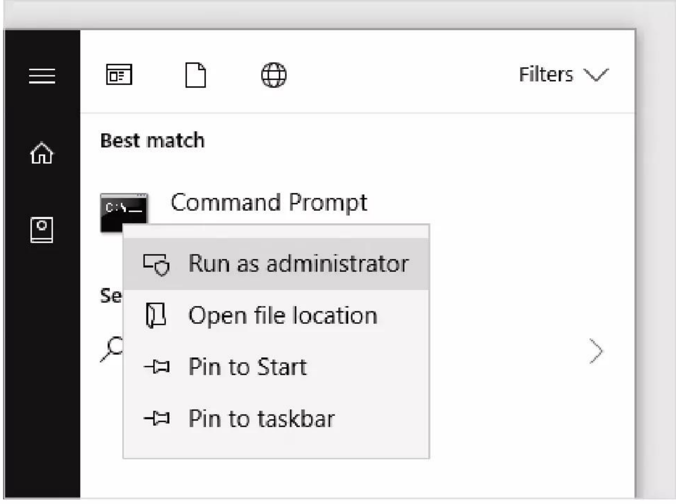
STEP 7

Depending on how your system is configured, you may not get any directory information at all or a message stating Access Denied. This is because the UAC is blocking access to protected areas of the system, like c:\Windows or C:\Program Files. Therefore, you need to run the batch file as an Administrator. Click the Windows Start button and enter **CMD** again.


Filters
Best match
Command Prompt
Desktop app
Search suggestions
cmd - See web results
cmd

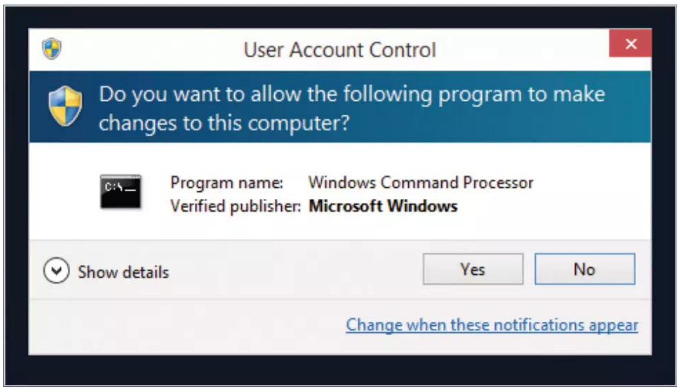
STEP 8

Instead of left clicking on the Command Prompt result, as you did the first time you opened it, right-click it and from the menu choose Run as Administrator. There is a risk that you could damage system files as the Administrator but as long as you're careful and don't do anything beyond viewing directories, you will be okay.


Filters
Best match
Command Prompt
Run as administrator
Open file location
Pin to Start
Pin to taskbar
Se

STEP 9

This action triggers the UAC warning message, asking you if you're sure you want to run the Windows command prompt with the elevated Administrator privileges. Most of the time we wouldn't recommend this course of action: the UAC is there to protect your system. In this case, however, click Yes.


User Account Control

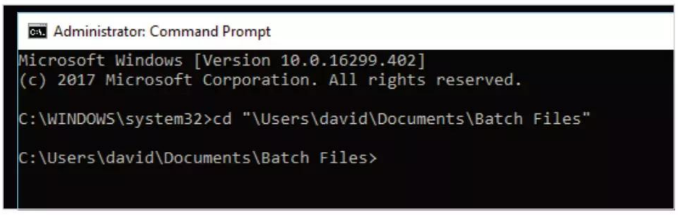
Do you want to allow the following program to make changes to this computer?

Program name: Windows Command Processor
Verified publisher: **Microsoft Windows**

Show details
Yes
No
Change when these notifications appear

STEP 10

With the UAC active, the command prompt looks a little different. For starters, it's now defaulting to the C:\WINDOWS\system32 folder and the top of the windows is labelled Administrator. To run the batch file, you need to navigate to the Batch Files directory with: **cd \Users\USERNAME\Documents\Batch Files**. To help, press the Tab key to auto-complete the directory names.


Administrator: Command Prompt

```
Microsoft Windows [Version 10.0.16299.402]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd "\Users\david\Documents\Batch Files"

C:\Users\david\Documents\Batch Files>
```

Batch File Programming

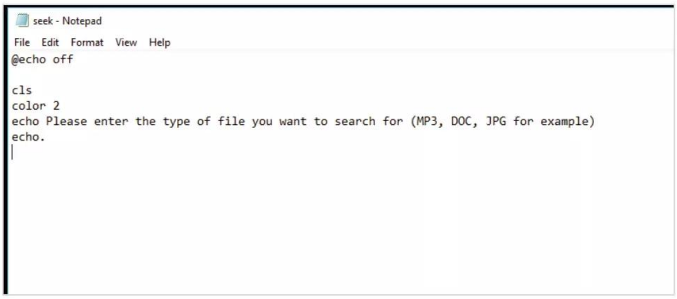
It's the little additions we can make to a batch file that help it stand out and ultimately become more useful. While the Windows graphical interface is still king, the command line can do just as much, and this is where batch files come into their own.

SEARCHING FOR FILES

Here's an interesting little batch file that you can easily extend for your own use. It asks the user for a file type to search for and displays the results.

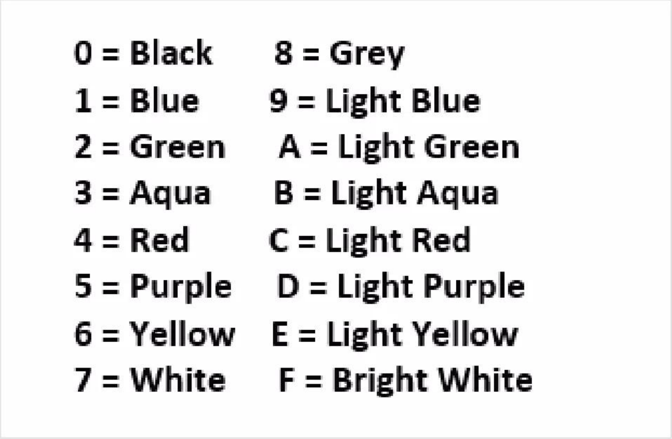
STEP 1 We are introducing a couple of new commands into the mix here but we think they're really useful. Create a new batch file called seek.bat and in it put:

```
@echo off
cls
color 2
echo Please enter the type of file you want to
search for (MP3, DOC, JPG for example)
echo.
```



```
@echo off

cls
color 2
echo Please enter the type of file you want to search for (MP3, DOC, JPG for example)
echo.
```

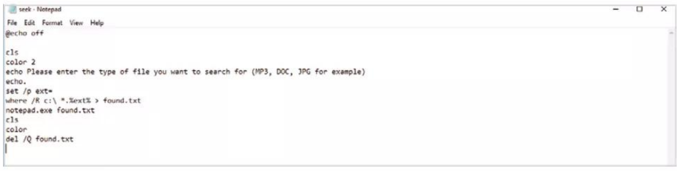
STEP 2 The new command in this instance is color (Americanised spelling). Color, as you already assume, changes the colour of the command prompt display. The color attributes are specified by two hex digits, the first corresponds to the background colour of the Command console and the second to the foreground, and can be any of the following values:


0 = Black
1 = Blue
2 = Green
3 = Aqua
4 = Red
5 = Purple
6 = Yellow
7 = White

8 = Grey
9 = Light Blue
A = Light Green
B = Light Aqua
C = Light Red
D = Light Purple
E = Light Yellow
F = Bright White

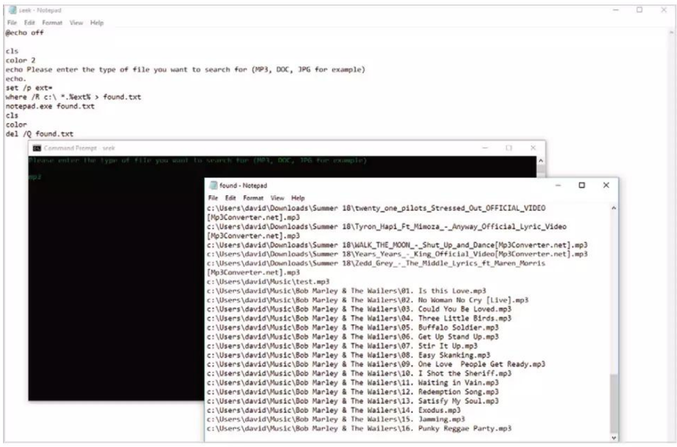
STEP 3 Now let's extend the seek.bat batch file:

```
@echo off
cls
color 2
echo Please enter the type of file you want to
search for (MP3, DOC, JPG for example)
echo.
set /p ext=
where /R c:\ *.%ext% > found.txt
notepad.exe found.txt
cls
color
del /Q found.txt
```



```
@echo off

cls
color 2
echo Please enter the type of file you want to search for (MP3, DOC, JPG for example)
echo.
set /p ext=
where /R c:\ *.%ext% > found.txt
notepad.exe found.txt
cls
color
del /Q found.txt
```

STEP 4 Another new command, Where, looks for a specific file or directory based on the user's requirements. In this case, we have created a blank variable called ext that the user can enter the file type in, which then searches using Where and dumps the results in a text file called found.txt. Save and run the batch file.



```
@echo off

cls
color 2
echo Please enter the type of file you want to search for (MP3, DOC, JPG for example)
echo.
set /p ext=
where /R c:\ *.%ext% > found.txt
notepad.exe found.txt
cls
color
del /Q found.txt
```

```
Please enter the type of file you want to search for (MP3, DOC, JPG for example)

mp3
```

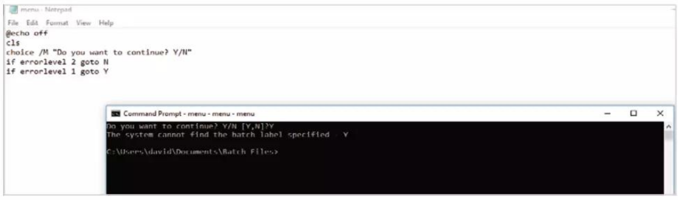
```
c:\Users\david\Downloads\Summer 18\twenty_one_pilots_Stressed_Out_OFFICIAL_VIDEO
[Mp3Converter.net].mp3
c:\Users\david\Downloads\Summer 18\Tyron_Hapi_Ft_Mimoza_-_Anyway_Official_Lyric_Video
[Mp3Converter.net].mp3
c:\Users\david\Downloads\Summer 18\WALK_THE_MOON_-_Shut_Up_and_Dance[Mp3Converter.net].mp3
c:\Users\david\Downloads\Summer 18\Years_Years_-_King_Official_Video[Mp3Converter.net].mp3
c:\Users\david\Downloads\Summer 18\Zedd_Grey_-_The_Middle_Lyrics_ft_Maren_Morris
[Mp3Converter.net].mp3
c:\Users\david\Music\test.mp3
c:\Users\david\Music\Bob Marley & The Wailers\01. Is this Love.mp3
c:\Users\david\Music\Bob Marley & The Wailers\02. No Woman No Cry [Live].mp3
c:\Users\david\Music\Bob Marley & The Wailers\03. Could You Be Loved.mp3
c:\Users\david\Music\Bob Marley & The Wailers\04. Three Little Birds.mp3
c:\Users\david\Music\Bob Marley & The Wailers\05. Buffalo Soldier.mp3
c:\Users\david\Music\Bob Marley & The Wailers\06. Get Up Stand Up.mp3
c:\Users\david\Music\Bob Marley & The Wailers\07. Stir It Up.mp3
c:\Users\david\Music\Bob Marley & The Wailers\08. Easy Skanking.mp3
c:\Users\david\Music\Bob Marley & The Wailers\09. One Love  People Get Ready.mp3
c:\Users\david\Music\Bob Marley & The Wailers\10. I Shot the Sheriff.mp3
c:\Users\david\Music\Bob Marley & The Wailers\11. Waiting in Vain.mp3
c:\Users\david\Music\Bob Marley & The Wailers\12. Redemption Song.mp3
c:\Users\david\Music\Bob Marley & The Wailers\13. Satisfy My Soul.mp3
c:\Users\david\Music\Bob Marley & The Wailers\14. Exodus.mp3
c:\Users\david\Music\Bob Marley & The Wailers\15. Jamming.mp3
c:\Users\david\Music\Bob Marley & The Wailers\16. Punky Reggae Party.mp3
```

CHOICE MENUS

Creating a menu of choices is a classic batch file use and a good example to help expand your batch file programming skills. Here's some code to help you understand how it all works.

STEP 1 Rather than using a variable to process a user's response, batch files can instead use the Choice command in conjunction with an ErrorLevel parameter to make a menu. Create a new file called menu.bat and enter the following:

```
@echo off
cls
choice /M “Do you want to continue? Y/N”
if errorlevel 2 goto N
if errorlevel 1 goto Y
goto End:
```

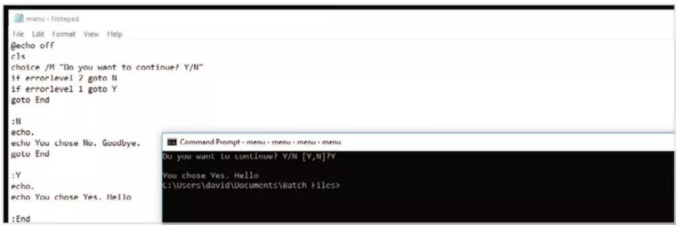

menu - Notepad
File Edit Format View Help
Command Prompt - menu - menu - menu

```
@echo off
cls
choice /M "Do you want to continue? Y/N"
if errorlevel 2 goto N
if errorlevel 1 goto Y
```

```
Do you want to continue? Y/N [Y,N]?Y
The system cannot find the batch label specified - Y

C:\Users\david\Documents\Batch Files>
```

STEP 2 Running the code produces an error as we've called a Goto command without any reference to it in the file. Goto does exactly that, goes to a specific line in the batch file. Finish the file with the following and run it again:

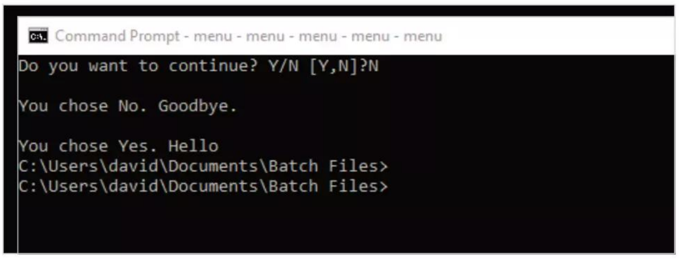
```
:N
echo.
echo You chose No. Goodbye.
goto End

:Y
echo.
echo You chose Yes. Hello

:End
```


menu - Notepad
File Edit Format View Help
Command Prompt - menu - menu - menu - menu

```
@echo off
cls
choice /M "Do you want to continue? Y/N"
if errorlevel 2 goto N
if errorlevel 1 goto Y
goto End

:N
echo.
echo You chose No. Goodbye.
goto End

:Y
echo.
echo You chose Yes. Hello

:End
```

```
Do you want to continue? Y/N [Y,N]?Y

You chose Yes. Hello
C:\Users\david\Documents\Batch Files>
```

STEP 3 The output from your choice is different depending on whether you pick Y or N. The :End part simply signifies the end of the file (also known as EOF). Without it the batch file runs through each line and display the Y response even if you enter N; so it's important to remember to follow your Goto commands.



```
Do you want to continue? Y/N [Y,N]?N

You chose No. Goodbye.

You chose Yes. Hello
C:\Users\david\Documents\Batch Files>
C:\Users\david\Documents\Batch Files>
```

STEP 4 ErrorLevels are essentially variables and the /M switch of Choice allows a descriptive message string to be displayed. Extend this menu with something new:

```
@echo off
cls
echo.
echo ---------------------------------------------
echo.
echo Please choose a directory.
echo.
echo Press 1 for c:\Music
echo.
echo Press 2 for c:\Documents
echo.
echo Press 3 for c:\Pictures
echo.
echo Press 4 for c:\Videos
echo.
echo ---------------------------------------------
choice /C 1234
if errorlevel 4 goto Videos
if errorlevel 3 goto Pictures
if errorlevel 2 goto Documents
if errorlevel 1 goto Music
```

STEP 5 Now add the Goto sections:

```
:Videos
cls
CD %HOMEPATH%\Videos
echo You are now in the Videos directory.
goto End

:Pictures
cls
CD %HOMEPATH%\Pictures
echo You are now in the Pictures directory.
goto End

:Documents
cls
CD %HOMEPATH%\Documents
echo You are now in the Documents directory.
goto End

:Music
cls
CD %HOMEPATH%\Music
echo you are now in the Music directory.
goto End

:End
```

STEP 6 When executed, the batch file displays a menu and with each choice the code changes directory to the one the user entered. The %HOMEPATH% system variable will enter the currently logged in user's Music, Pictures and so directories, and not anyone else's.

Loops and Repetition

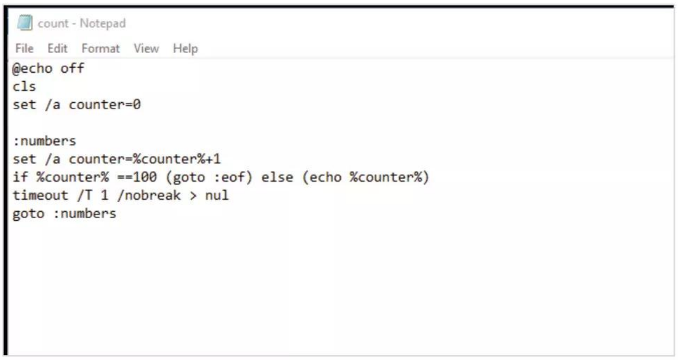
Looping and repeating commands are the staple diet of every programming language, including batch file programming. For example, you can create a simple countdown or even make numbered files or directories in the system.

COUNTERS

Creating code that counts in increasing or decreasing number sets is great for demonstrating loops. With that in mind, let's look at the If statement a little more, alongside more variables, and introduce the Else, Timeout and eof (End of File) commands.

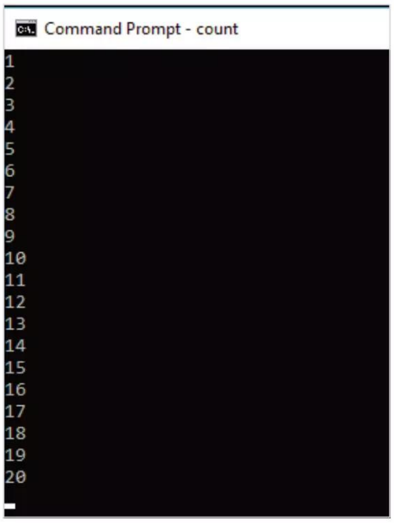
STEP 1 Start by creating a new batch file called count.bat. Enter the following, save it and run:

```
@echo off
cls
set /a counter=0

:numbers
set /a counter=%counter%+1
if %counter% ==100 (goto :eof) else (echo 
%counter%)
timeout /T 1 /nobreak > nul
goto :numbers
```



```
@echo off
cls
set /a counter=0

:numbers
set /a counter=%counter%+1
if %counter% ==100 (goto :eof) else (echo %counter%)
timeout /T 1 /nobreak > nul
goto :numbers
```

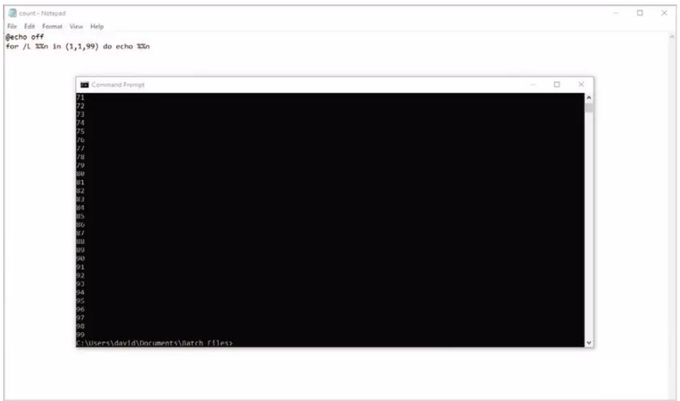
STEP 2 The count.bat code starts at number one and counts, scrolling down the screen, until it reaches 100. The Timeout command leaves a one second gap between numbers and the Else statement continues until the counter variable equals 100 before going to the eof (End Of File), thus closing the loop.



```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

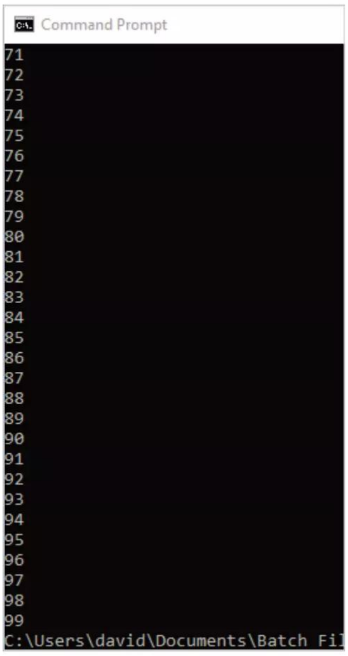
STEP 3 The count.bat is a rough way of demonstrating a loop; a better approach would be to use a for loop. Try this example instead:

```
@echo off
for /L %%n in (1,1,99) do echo %%n
```



```
@echo off
for /L %%n in (1,1,99) do echo %%n
```

STEP 4 Breaking it down, there's For, then the /L switch, which handles a range of numbers. Then the parameter labelled as %%n to denote a number. Then the in (1,1,99) part, which tells the statement how to count, as in 1 (start number), 1 (steps to take), 99 (the end number). The next part is do, meaning DO whatever command is after.

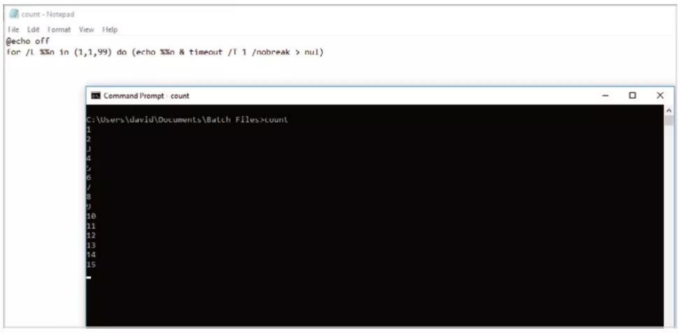


```
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
C:\Users\david\Documents\Batch Fil
```

STEP 5

You can include the pause between the numbers easily enough within the far simpler For loop by adding multiple commands after the Do For loop. The brackets and ampersand (&) separate the different commands. Try this:

```
@echo off
for /L %%n in (1,1,99) do (echo %%n & timeout /T 1
/nobreak > nul)
```


count - Notepad
File Edit Format View Help
Command Prompt - count

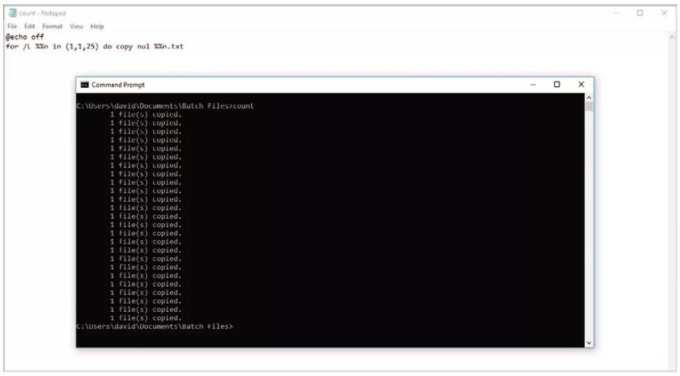
```
@echo off
for /L %%n in (1,1,99) do (echo %%n & timeout /T 1 /nobreak > nul)
```

```
C:\Users\david\Documents\Batch Files>count
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

STEP 6

One of the great time saving uses of batch files is to create multiple, numbered files. Assume that you want twenty five text files within a directory, all numbered from 1 to 25. A For loop much like the previous example does the trick:

```
@echo off
for /L %%n in (1,1,25) do copy nul %%n.txt
```


count - Notepad
File Edit Format View Help
Command Prompt

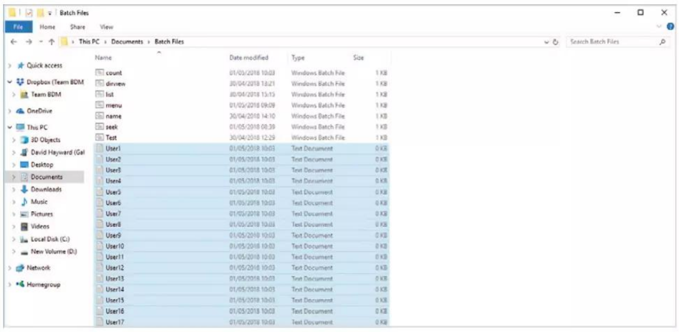
```
@echo off
for /L %%n in (1,1,25) do copy nul %%n.txt
```

```
C:\Users\david\Documents\Batch Files>count
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
C:\Users\david\Documents\Batch Files>
```

STEP 7

If you open Windows Explorer, and navigate to the Batch Files directory where you're working from, you can now see 25 text files all neatly numbered. Of course, you can append the file name with something like user1.txt and so on by altering the code to read:

```
@echo off
for /L %%n in (1,1,25) do copy nul User%%n.txt
```

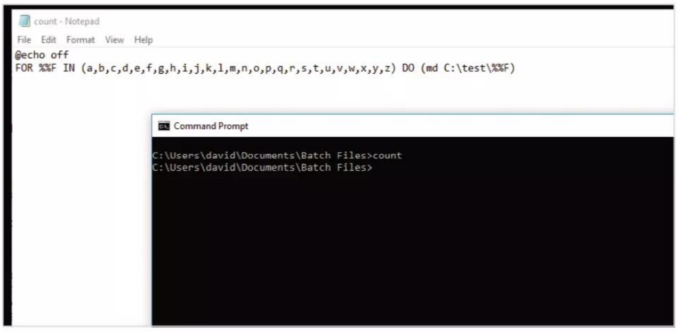

Batch Files
File
Home
Share
View
This PC › Documents › Batch Files
Search Batch Files
Quick access
Dropbox (Team BDM
Team BDM
OneDrive
This PC
3D Objects
David Hayward (Gal
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
New Volume (D:)
Network
Homegroup

Name	Date modified	Type	Size
count	01/05/2018 10:03	Windows Batch File	1 KB
dirview	30/04/2018 13:21	Windows Batch File	1 KB
list	30/04/2018 15:15	Windows Batch File	1 KB
menu	01/05/2018 09:09	Windows Batch File	1 KB
name	30/04/2018 14:10	Windows Batch File	1 KB
seek	01/05/2018 08:39	Windows Batch File	1 KB
Test	30/04/2018 12:29	Windows Batch File	1 KB
User1	01/05/2018 10:03	Text Document	0 KB
User2	01/05/2018 10:03	Text Document	0 KB
User3	01/05/2018 10:03	Text Document	0 KB
User4	01/05/2018 10:03	Text Document	0 KB
User5	01/05/2018 10:03	Text Document	0 KB
User6	01/05/2018 10:03	Text Document	0 KB
User7	01/05/2018 10:03	Text Document	0 KB
User8	01/05/2018 10:03	Text Document	0 KB
User9	01/05/2018 10:03	Text Document	0 KB
User10	01/05/2018 10:03	Text Document	0 KB
User11	01/05/2018 10:03	Text Document	0 KB
User12	01/05/2018 10:03	Text Document	0 KB
User13	01/05/2018 10:03	Text Document	0 KB
User14	01/05/2018 10:03	Text Document	0 KB
User15	01/05/2018 10:03	Text Document	0 KB
User16	01/05/2018 10:03	Text Document	0 KB
User17	01/05/2018 10:03	Text Document	0 KB

STEP 8

There are different ways of using the For loop. In this example, the code creates 26 directories, one for each letter of the alphabet, within the directory c:\test which the batch file makes using the MD command:

```
@echo off
FOR %%F IN (a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,
s,t,u,v,w,x,y,z) DO (md C:\test\%%F)
```

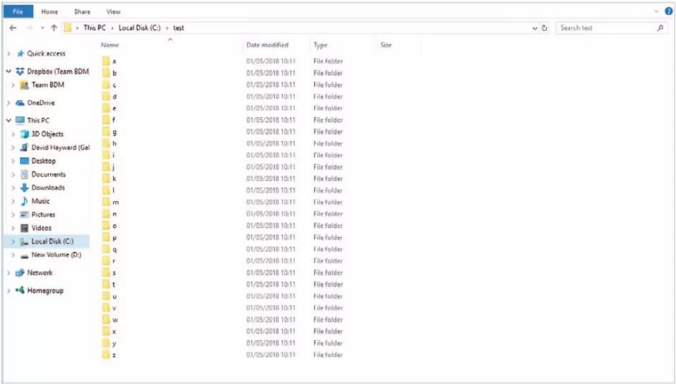

count - Notepad
File Edit Format View Help
Command Prompt

```
@echo off
FOR %%F IN (a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z) DO (md C:\test\%%F)
```

```
C:\Users\david\Documents\Batch Files>count
C:\Users\david\Documents\Batch Files>
```

STEP 9

Loops can be powerful and extremely useful elements in a batch file. While creating 26 directories may not sound too helpful, imagine having to create 1,000 users on a network and assign each one their own set of unique directories. This is where a batch file saves an immense amount of time.

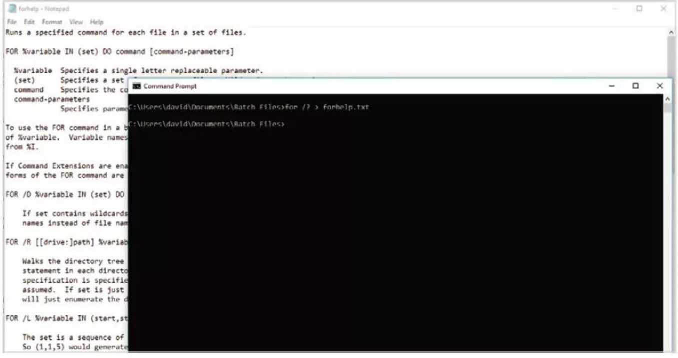

File
Home
Share
View
This PC › Local Disk (C:) › test
Search test
Quick access
Dropbox (Team BDM
Team BDM
OneDrive
This PC
3D Objects
David Hayward (Gal
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
New Volume (D:)
Network
Homegroup

Name	Date modified	Type	Size
a	01/05/2018 10:11	File folder	
b	01/05/2018 10:11	File folder	
c	01/05/2018 10:11	File folder	
d	01/05/2018 10:11	File folder	
e	01/05/2018 10:11	File folder	
f	01/05/2018 10:11	File folder	
g	01/05/2018 10:11	File folder	
h	01/05/2018 10:11	File folder	
i	01/05/2018 10:11	File folder	
j	01/05/2018 10:11	File folder	
k	01/05/2018 10:11	File folder	
l	01/05/2018 10:11	File folder	
m	01/05/2018 10:11	File folder	
n	01/05/2018 10:11	File folder	
o	01/05/2018 10:11	File folder	
p	01/05/2018 10:11	File folder	
q	01/05/2018 10:11	File folder	
r	01/05/2018 10:11	File folder	
s	01/05/2018 10:11	File folder	
t	01/05/2018 10:11	File folder	
u	01/05/2018 10:11	File folder	
v	01/05/2018 10:11	File folder	
w	01/05/2018 10:11	File folder	
x	01/05/2018 10:11	File folder	
y	01/05/2018 10:11	File folder	
z	01/05/2018 10:11	File folder	

STEP 10

Should you ever get stuck when using the various commands within a batch file, drop into the command prompt and enter the command followed by a question switch. For example, for /? or if /?. You get an on-screen help file detailing the commands' use. For easier reading, pipe it to a text file:

```
For /? > forhelp.txt
```


forhelp - Notepad
File Edit Format View Help
Command Prompt

```
Runs a specified command for each file in a set of files.

FOR %variable IN (set) DO command [command-parameters]

  %variable  Specifies a single letter replaceable parameter.
  (set)      Specifies a set
  command    Specifies the co
  command-parameters
             Specifies parame

To use the FOR command in a b
of %variable.  Variable names
from %I.

If Command Extensions are ena
forms of the FOR command are

FOR /D %variable IN (set) DO

    If set contains wildcards
    names instead of file nam

FOR /R [[drive:]path] %variab

    Walks the directory tree
    statement in each directo
    specification is specifie
    assumed.  If set is just
    will just enumerate the d

FOR /L %variable IN (start,st

    The set is a sequence of
    So (1,1,5) would generate
```

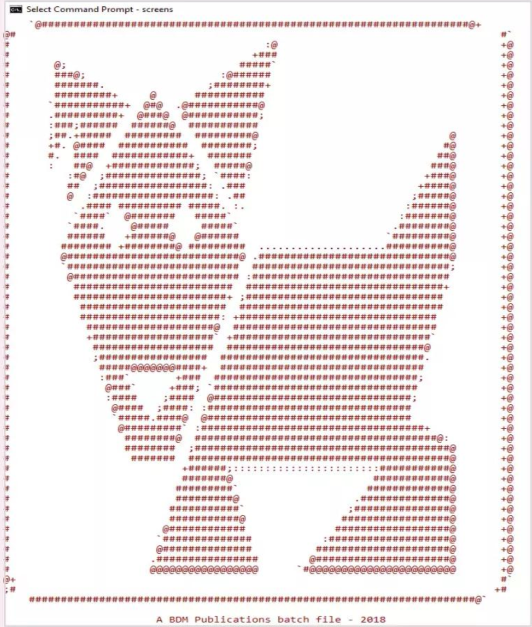
```
C:\Users\david\Documents\Batch Files>for /? > forhelp.txt

C:\Users\david\Documents\Batch Files>
```

Creating a Batch File Game

Based on what you've looked at so far with batch files, you can probably come up with your own simple text adventure or multiple-choice game. Here's one that we created, with which you're free to edit and make your own.

Make up your own questions but how about also including an introductory or loading screen? Make your loading screen in a separate batch file and save it as screens.bat (for example). Then, from the main game batch file, you can load it at the beginning of the file with the 'call' command followed by colour to reset the game's colours:

```
@echo off
Cls
Call screens.bat
color
:start
set /a score=0
set /a question=0
cls
set /p name= What is your name?
```



Select Command Prompt - screens

A BDM Publications batch file - 2018

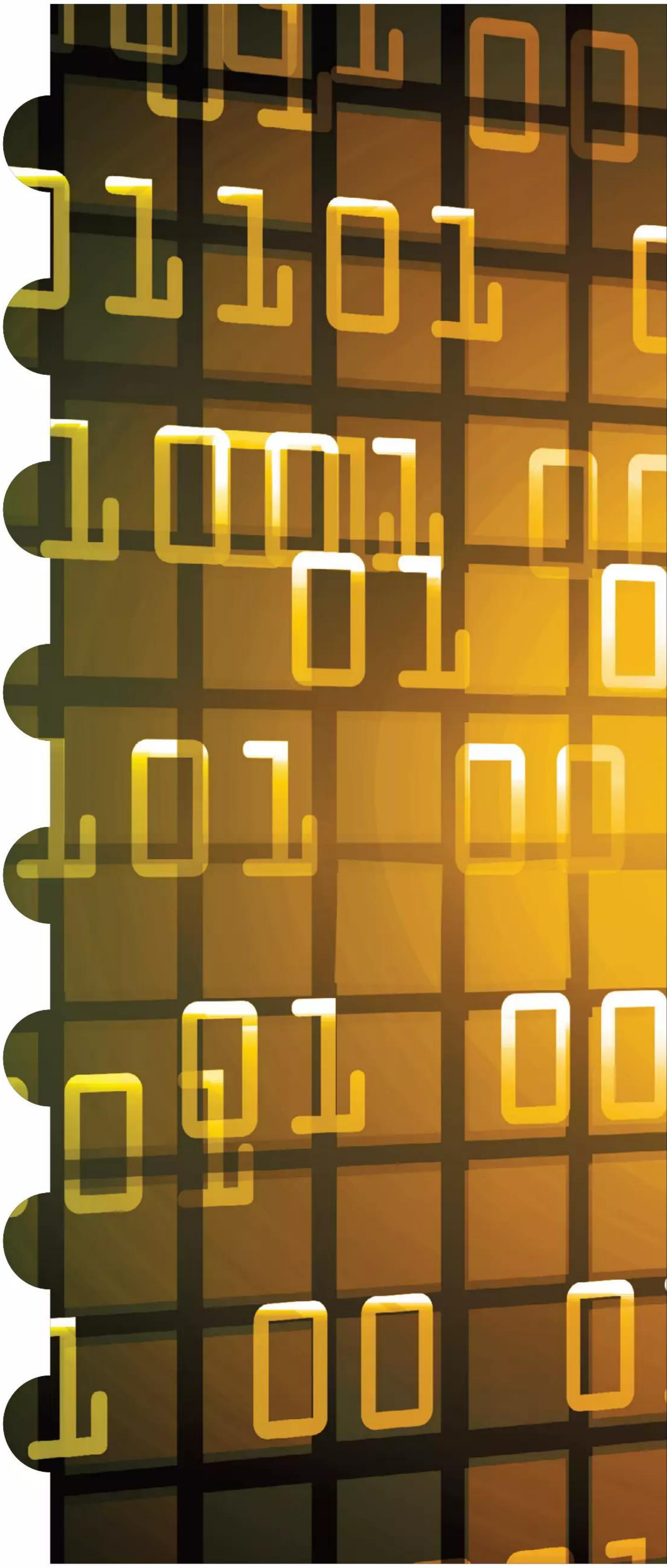
```
@echo off
Cls
:start
set /a score=0
set /a question=0
cls
set /p name= What is your name?

:menu
cls
echo.
echo ***********************************************
   **
echo.
echo Welcome %name% to the super-cool trivia game.
echo.
echo Press 1 to get started
echo.
echo Press 2 for instructions
echo.
echo Press Q to quit
echo.
echo ***********************************************
   **
choice /C 12Q
if errorlevel 3 goto :eof
if errorlevel 2 goto Instructions
if errorlevel 1 goto Game

:Instructions
cls
echo.
echo **********************************
echo.
echo The instructions are simple. Answer the
   questions correctly.
echo.
echo **********************************
pause
cls
goto menu

:Game
set /a question=%question%+1
cls
if %question% ==5 (goto end) else (echo you are on
   question %question%)
echo.
echo get ready for the question...
echo.
timeout /T 5 /nobreak > nul
if %question% ==5 (goto end) else (goto %question%)

:1
cls
echo.
echo ***************************
echo.
```

```
echo Your current score is %score%
echo.
echo ****************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following version of Windows is the
   best?
echo.
echo A. Windows 10
echo.
echo B. Windows ME
echo.
echo C. Windows Vista
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:2
cls
echo.
echo ****************************
echo.
echo Your current score is %score%
echo.
echo ****************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following version of Windows is the
   most stable?
echo.
echo A. Windows 10
echo.
echo B. Windows 95
echo.
echo C. Windows ME
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:3
cls
echo.
echo ****************************
echo.
echo Your current score is %score%
echo.
echo ****************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following Windows version is the
   latest?
echo.
echo A. Windows 10
echo.
echo B. Windows 98
echo.
echo C. Windows 7
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:4
cls
echo.
echo ****************************
echo.
echo Your current score is %score%
echo.
echo ****************************
echo.
echo.
echo Question %question%.
echo.
echo Which of the following Windows uses DirectX 12?
echo.
echo A. Windows 10
echo.
echo B. Windows 3.11
echo.
echo C. Windows XP
echo.
choice /C abc
if errorlevel 3 goto wrong
if errorlevel 2 goto wrong
if errorlevel 1 goto correct

:Wrong
cls
echo *****************
echo.
echo WRONG!!!!
echo.
echo *****************
set /a score=%score%-1
pause
goto :game

:correct
cls
echo *****************
echo.
echo CORRECT. YIPEE!!!
echo.
echo *****************
set /a score=%score%+1
pause
goto :game

:end
cls
echo ****************************
echo.
echo Well done, %name%, you have answered all the
   questions
echo.
echo And your final score is....
echo.
echo %score%
echo.
echo ******************************
choice /M “play again? Y/N”
if errorlevel 2 goto :eof
if errorlevel 1 goto start
```

Linux is such a versatile operating system that's both malleable and powerful, while offering the programmer a perfect foundation on which to build their skills. While all the popular and mainstream programming languages are available on Linux, as they are on Windows and macOS, Linux also utilises its own coding language, called scripting.

Bash scripting on Linux can be used to create a wealth of useful, real-world programs that interact with the user, or simply work in the background based on a pre-defined schedule. Scripting is a powerful interface to the Linux system, so we've crafted this section to help you get to grips with how everything fits together, and how to make some amazing Linux scripts.

112 Why Linux?
114 The Best Linux Distributions
116 Equipment You Will Need
118 Creating a Linux Installer on Windows
120 Installing Linux on a PC
122 Installing a Virtual Environment
124 Installing Linux in a Virtual Environment
126 Getting Ready to Code in Linux
128 Creating Bash Scripts – Part 1
130 Creating Bash Scripts – Part 2
132 Creating Bash Scripts – Part 3
134 Creating Bash Scripts – Part 4
136 Creating Bash Scripts – Part 5
138 Command Line Quick Reference



Coding on Linux

Why Linux?

For many of its users, Linux means freedom. Freedom from the walled-gardened approach of other operating systems, freedom to change and use the OS as you please and freedom from any form of licensing or payment. There's a lot more to Linux than you may think.

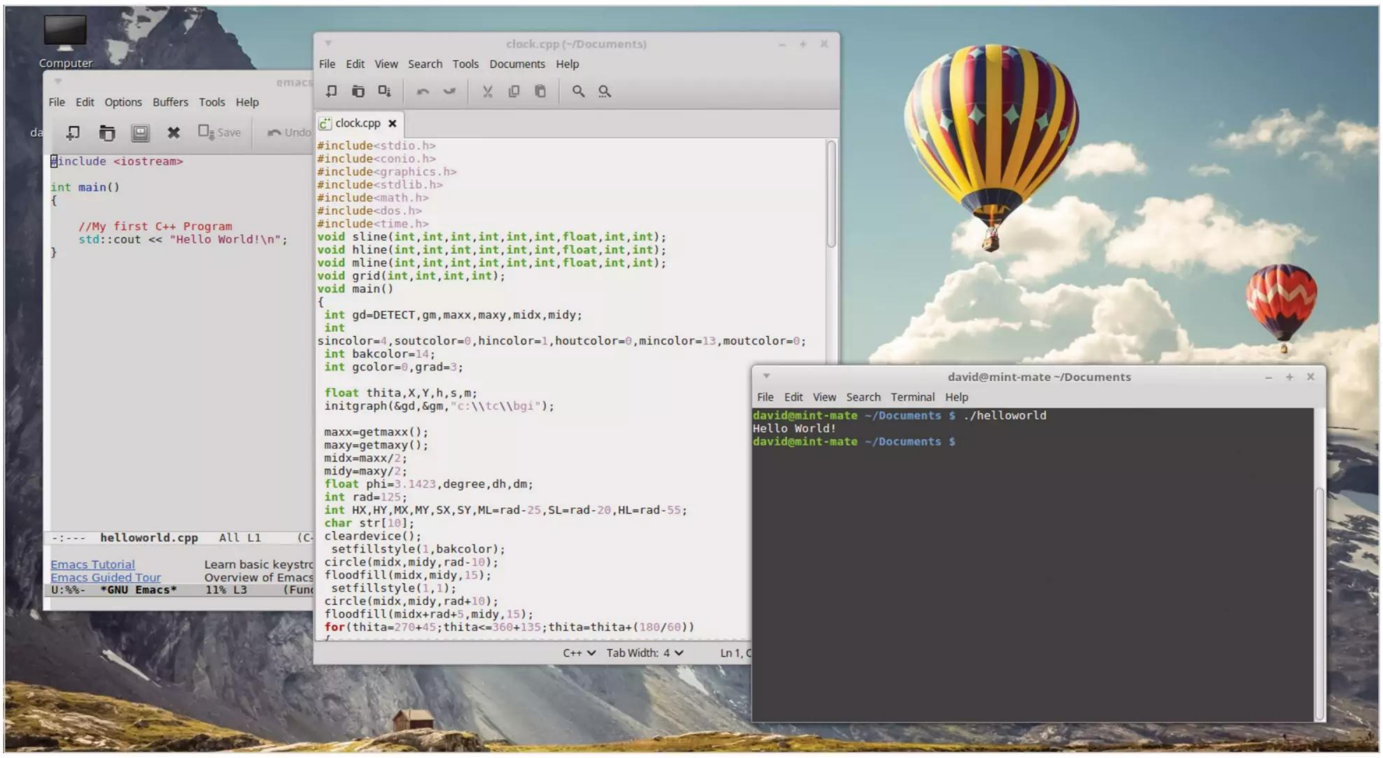
FREE AND OPEN

Linux is a fantastic fit for those who want something different; the efficiency of the system, the availability of applications and its stability, are just a few reasons why.

The first thing we need to address is that there is no basic operating system called Linux. Fundamentally, Linux is the operating system kernel, the core component of an OS. When talking about Linux we are, in fact, referring to one of the many distributions, or distros, that use the Linux kernel. No doubt you've heard of at least one of the current popular distros: Ubuntu, Linux Mint, Fedora, openSUSE, Debian, Raspbian… the list goes on. Each one of these distros offers the user something a little different. While each has the Linux kernel at its core, they offer a different looking desktop environment, different pre-loaded applications, different ways in which to update the system and get more apps installed and a slightly different look and feel throughout the entire system. However, at the centre lies Linux, which is why we say, Linux.

Linux works very differently to Windows or macOS. For a start, it's free to download, free to install on as many computers as you like, free to use for an unlimited amount of time and free to upgrade and extend with equally free programs and applications. This free to use element is one of the biggest draws for the developer. While a Windows license can cost up to £100 and a Mac considerably more, a user, be they a developer, gamer, or someone who wants to put an older computer to use, can quickly download a distro and get to work in a matter of minutes.

Alongside the free to use aspect comes a level of freedom to customise and mould the system to your own uses. Each of the distros available on the Internet have a certain 'spin', some offer increased security, a fancy-


Computer
File Edit Options Buffers Tools Help
Save
Undo
-:--- helloworld.cpp All L1
Emacs Tutorial
Emacs Guided Tour
Learn basic keystro
Overview of Emacs
U:%%- *GNU Emacs* 11% L3 (Func
clock.cpp (~/Documents)
File Edit View Search Tools Documents Help
clock.cpp
C++ Tab Width: 4 Ln 1, C
david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>
#include<dos.h>
#include<time.h>
void sline(int,int,int,int,int,int,float,int,int);
void hline(int,int,int,int,int,int,float,int,int);
void mline(int,int,int,int,int,int,float,int,int);
void grid(int,int,int,int);
void main()
{
 int gd=DETECT,gm,maxx,maxy,midx,midy;
 int
sincolor=4,soutcolor=0,hincolor=1,houtcolor=0,mincolor=13,moutcolor=0;
 int bakcolor=14;
 int gcolor=0,grad=3;

 float thita,X,Y,h,s,m;
 initgraph(&gd,&gm,"c:\\tc\\bgi");

 maxx=getmaxx();
 maxy=getmaxy();
 midx=maxx/2;
 midy=maxy/2;
 float phi=3.1423,degree,dh,dm;
 int rad=125;
 int HX,HY,MX,MY,SX,SY,ML=rad-25,SL=rad-20,HL=rad-55;
 char str[10];
 cleardevice();
  setfillstyle(1,bakcolor);
 circle(midx,midy,rad-10);
 floodfill(midx,midy,15);
  setfillstyle(1,1);
 circle(midx,midy,rad+10);
 floodfill(midx+rad+5,midy,15);
 for(thita=270+45;thita<=360+135;thita=thita+(180/60))
```

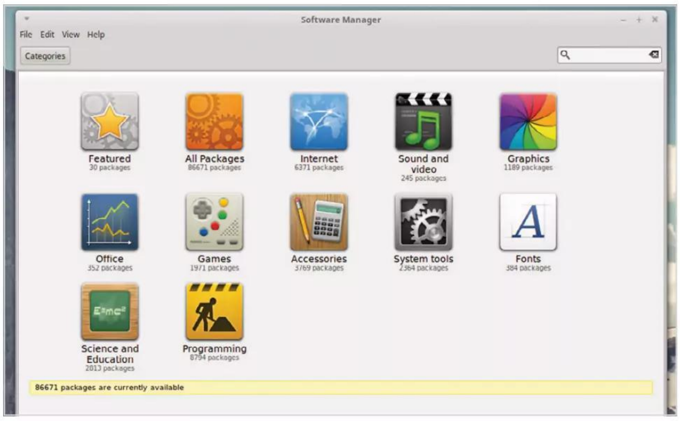
```
david@mint-mate ~/Documents $ ./helloworld
Hello World!
david@mint-mate ~/Documents $
```

Linux is a great operating system in which to start coding.

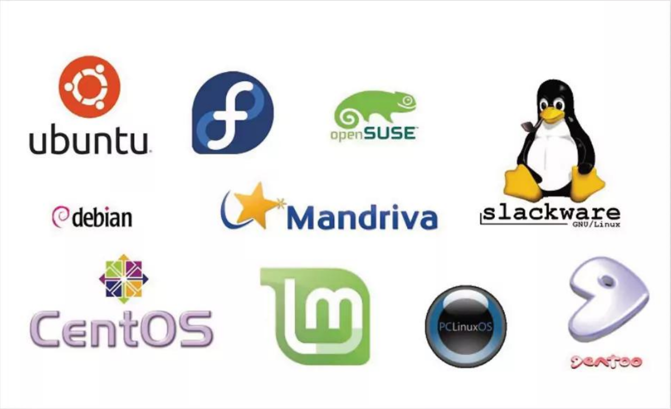
looking desktop, a gaming specific spin, or something directed toward students. This extensibility makes Linux a more desirable platform to use, as you can quickly mould the system into a development base, including many different kinds of IDEs for use with Python, web development, C++, Java and so on, or even a base for online anonymity, perhaps as a Minecraft server, media centre and much more.

Another remarkable advantage for those looking to learn how to code is that Linux comes with most of the popular coding environments built-in. Both Python and C++ are pre-installed in a high percentage of Linux distros, which means you can start to program almost as soon as you install the system and boot it up for the first time.

Generally speaking, Linux doesn't take up as many system resources as Windows or macOS, (by system resources, we mean memory, hard drive space and CPU load), as the Linux code has been streamlined and is free from third-party 'bloatware', which hogs those systems resources. A more efficient system, means more available resources for the coding and testing environment and the programs you will eventually create. Less resource use also means you can use Linux on older hardware that would normally struggle, or even refuse to run, the latest versions of Windows or macOS.


Software Manager
File Edit View Help
Categories
Featured
30 packages
All Packages
86671 packages
Internet
6371 packages
Sound and video
245 packages
Graphics
1189 packages
Office
352 packages
Games
1971 packages
Accessories
3769 packages
System tools
2364 packages
Fonts
384 packages
Science and Education
2013 packages
Programming
8794 packages
86671 packages are currently available

There are thousands of free packages available for programmers under Linux.


ubuntu
openSUSE
debian
Mandriva
slackware GNU/Linux
CentOS
PCLinuxOS
gentoo

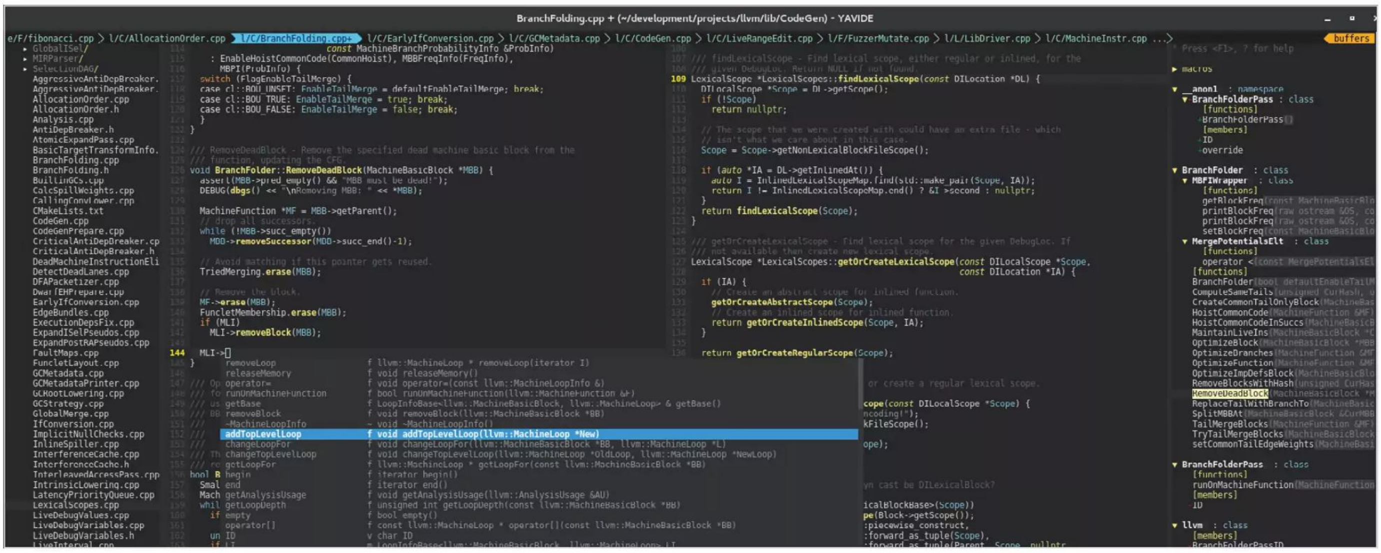
Each distro offers something unique to the user but all have Linux at the core.

So rather than throwing away an old computer, it can be reused with a Linux distro.

However, it's not all about C++, Python, or any of the other more popular programming languages. Using the command line of Linux (also called the Terminal), you're able to create Shell scripts, which are programs made up of scripting languages and designed to run from the command line. They are used mainly to automate tasks, or offer the user some form of input and output for a certain operation.

There are many more advantages we could list, for example, there are thousands and thousands of free programs and apps available that cover practically every aspect of computing. Known as packages, there are (at the time of writing) over 8,700 specific programming applications on Linux Mint alone and an incredible 62,000+ overall packages, catering from Amateur Radio to WWW tools.

In conclusion, Linux is a great resource and environment in which to program. It's perfectly suited for developers, while continually improving and evolving. If you're serious about getting into coding, or you just want to explore something new, then give Linux a try and see how it works for you.


BranchFolding.cpp + (~/development/projects/llvm/lib/CodeGen) - YAVIDE
l/C/AllocationOrder.cpp > l/C/BranchFolding.cpp+ > l/C/EarlyIfConversion.cpp > l/C/GCMetadata.cpp > l/C/CodeGen.cpp > l/C/LiveRangeEdit.cpp > l/F/FuzzerMutate.cpp > l/L/LibDriver.cpp > l/C/MachineInstr.cpp ...>
buffers

```
                              const MachineBranchProbabilityInfo &ProbInfo)
    : EnableHoistCommonCode(CommonHoist), MBBFreqInfo(FreqInfo),
      MBPI(ProbInfo) {
  switch (FlagEnableTailMerge) {
  case cl::BOU_UNSET: EnableTailMerge = defaultEnableTailMerge; break;
  case cl::BOU_TRUE: EnableTailMerge = true; break;
  case cl::BOU_FALSE: EnableTailMerge = false; break;
  }
}

/// RemoveDeadBlock - Remove the specified dead machine basic block from the
/// function, updating the CFG.
void BranchFolder::RemoveDeadBlock(MachineBasicBlock *MBB) {
  assert(MBB->pred_empty() && "MBB must be dead!");
  DEBUG(dbgs() << "\nRemoving MBB: " << *MBB);

  MachineFunction *MF = MBB->getParent();
  // drop all successors.
  while (!MBB->succ_empty())
    MBB->removeSuccessor(MBB->succ_end()-1);

  // Avoid matching if this pointer gets reused.
  TriedMerging.erase(MBB);

  // Remove the block.
  MF->erase(MBB);
  FuncletMembership.erase(MBB);
  if (MLI)
    MLI->removeBlock(MBB);

  MLI->
}
```

```
removeLoop              f llvm::MachineLoop * removeLoop(iterator I)
releaseMemory           f void releaseMemory()
operator=               f void operator=(const llvm::MachineLoopInfo &)
runOnMachineFunction    f bool runOnMachineFunction(llvm::MachineFunction &F)
getBase                 f LoopInfoBase<llvm::MachineBasicBlock, llvm::MachineLoop> & getBase()
removeBlock             f void removeBlock(llvm::MachineBasicBlock *BB)
~MachineLoopInfo        ~ void ~MachineLoopInfo()
addTopLevelLoop         f void addTopLevelLoop(llvm::MachineLoop *New)
changeLoopFor           f void changeLoopFor(llvm::MachineBasicBlock *BB, llvm::MachineLoop *L)
changeTopLevelLoop      f void changeTopLevelLoop(llvm::MachineLoop *OldLoop, llvm::MachineLoop *NewLoop)
getLoopFor              f llvm::MachineLoop * getLoopFor(const llvm::MachineBasicBlock *BB)
begin                   f iterator begin()
end                     f iterator end()
getAnalysisUsage        f void getAnalysisUsage(llvm::AnalysisUsage &AU)
getLoopDepth            f unsigned int getLoopDepth(const llvm::MachineBasicBlock *BB)
empty                   f bool empty()
operator[]              f const llvm::MachineLoop * operator[](const llvm::MachineBasicBlock *BB)
ID                      v char ID
```

```
/// findLexicalScope - Find lexical scope, either regular or inlined, for the
/// given DebugLoc. Return NULL if not found.
LexicalScope *LexicalScopes::findLexicalScope(const DILocation *DL) {
  DILocalScope *Scope = DL->getScope();
  if (!Scope)
    return nullptr;

  // The scope that we were created with could have an extra file - which
  // isn't what we care about in this case.
  Scope = Scope->getNonLexicalBlockFileScope();

  if (auto *IA = DL->getInlinedAt()) {
    auto I = InlinedLexicalScopeMap.find(std::make_pair(Scope, IA));
    return I != InlinedLexicalScopeMap.end() ? &I->second : nullptr;
  }
  return findLexicalScope(Scope);
}

/// getOrCreateLexicalScope - Find lexical scope for the given DebugLoc. If
/// not available then create new lexical scope.
LexicalScope *LexicalScopes::getOrCreateLexicalScope(const DILocalScope *Scope,
                                                     const DILocation *IA) {
  if (IA) {
    // Create an abstract scope for inlined function.
    getOrCreateAbstractScope(Scope);
    // Create an inlined scope for inlined function.
    return getOrCreateInlinedScope(Scope, IA);
  }

  return getOrCreateRegularScope(Scope);
```

A Linux programming environment can be as simple or as complex as you need it to be.

The Best Linux Distributions

There are numerous versions of Linux available, known as “Distributions”. Each has a different ethos and approach. Here are five great distributions to try and where you can get them.

LINUX MINT – LINUXMINT.COM

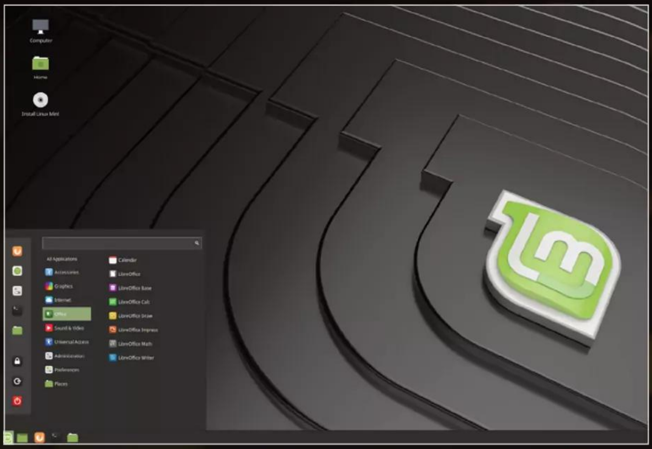
By far the most popular Linux distro (distribution) is Linux Mint. Mint began life back in 2006, as an alternative to the then, most popular distro, Ubuntu. Although based on Ubuntu’s Long Term Support build, Linux Mint took a different direction to offer the user a better overall experience.

Linux Mint has three main desktop versions available with each new version of the core OS it releases. This may sound confusing at first, but it’s quite simple. Currently, Linux Mint uses the Cinnamon Desktop Environment as its flagship model, although MATE and Xfce models are also available.

Cinnamon is a graphically rich desktop environment, while MATE uses less fancy graphics but is more stable on a wider variety of desktop systems and finally Xfce is an extremely streamlined desktop environment that is built for speed and ultimate stability.

Throughout this title, we’ll be using the Cinnamon version, however, you can try out any of the other desktop environments.

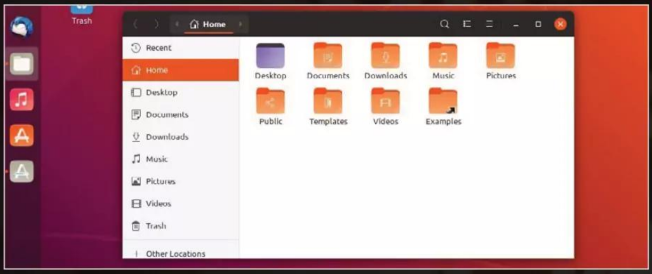
In fact, it’s recommended that you spend at least some time trying different environments and even different distros, to see which suits you and your computer best.



UBUNTU – UBUNTU.COM

The second most popular distro available is Ubuntu: an ancient African word meaning ‘humanity to others’. Ubuntu’s popularity has fluctuated over its fourteen-year life. At one time, it was easily the most used, Linux-based operating system in the world, sadly, some wrong choices along the way with regards to its presentation, along with some unfavourable and controversial elements involving privacy, saw it topple from the number one spot.

That said; Ubuntu has since made amends and is slowing crawling its way back up the Linux leader board. The latest versions of the OS use the GNOME 3 desktop environment, it’s impressive although it can be a little confusing for former Windows users and it’s a little heavy on system resources (especially if you plan to install it on an older computer).



Ubuntu, for all its faults, is a good Linux distro to start experimenting with. It’s a clean interface, easy to use and install and offers the user the complete Linux experience.

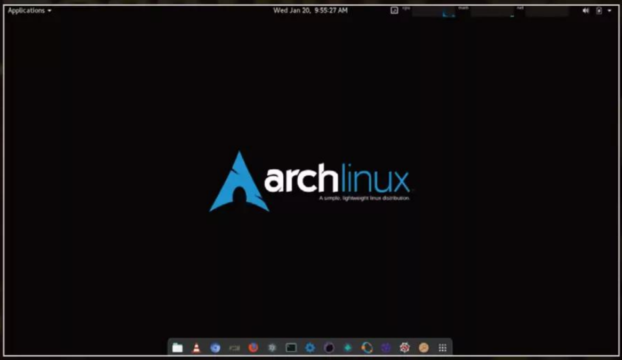
ARCH – ARCHLINUX.ORG

Arch is one of the longest running Linux distributions and forms the basis of many other versions of Linux. So why install Mint, or Ubuntu when you can install Arch? Many users do exactly that, but it's not ideal for beginners. Ubuntu and Mint both offer an easier installation path and come with software packages that will help you get started.

Arch, on the other hand, is a more 'bare bones' affair. It is committed to free software and its repositories contain over 50,000 apps to install. You can also install multiple Desktop environments and use them as you would with any other distro.

Arch is a distro for when you're more experienced with Linux. You start with nothing more than the command line and from there: you manually partition your hard drive, set where the installation files will go, create a user, set the OS locale and install a desktop environment, along with the apps you want.

The advantage, for all this hard work, is a distro that you've created. This means your Arch distro won't come with all the unnecessary files and apps that other distros have pre-installed; it's custom made for you, by you.

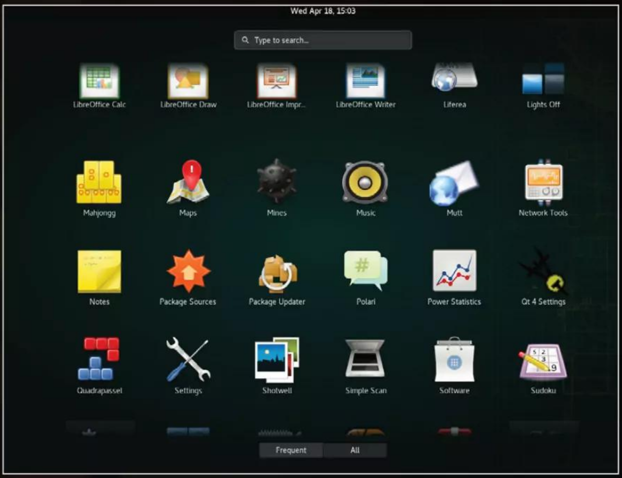

Applications
Wed Jan 20, 9:55:27 AM
archlinux
A simple, lightweight linux distribution.

OPENSUSE – OPENSUSE.ORG

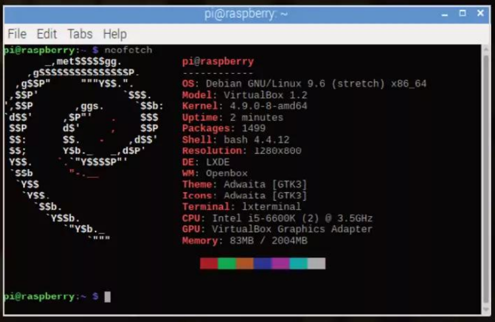
Most Linux distributions fall into two camps: those with the latest features and technology like Ubuntu and Mint and those with fewer new features, but rock-solid reliability, like Debian.

Meanwhile, openSUSE attempts to cover both bases. OpenSUSE Leap is a rock-solid system. It's developed openly by a community, along with SUSE employees. They develop an enterprise-level operating system: SUSE, which powers the London Stock Exchange, amongst other things. It is designed for mission critical environments where 'there is no scope for instability'. If you find all that too sensible, openSUSE Tumbleweed is a rolling release with all the latest features and the occasional crash.

openSUSE is a highly respected Linux distribution and many of its core contributors work on the Linux Kernel, LibreOffice, Gnome and other key Linux areas. In short: openSUSE is where you'll find the pros hanging out.


Wed Apr 18, 15:03
Type to search...
LibreOffice Calc
LibreOffice Draw
LibreOffice Impr...
LibreOffice Writer
Liferea
Lights Off
Mahjongg
Maps
Mines
Music
Mutt
Network Tools
Notes
Package Sources
Package Updater
Polari
Power Statistics
Qt 4 Settings
Quadrapassel
Settings
Shotwell
Simple Scan
Software
Sudoku
Frequent
All

RASPBERRY PI DESKTOP – RASPBERRYPI.ORG/DOWNLOADS/RASPBERRY-PI-DESKTOP


pi@raspberry: ~
File
Edit
Tabs
Help

```
pi@raspberry:~ $ neofetch
       _,met$$$$$gg.          pi@raspberry
    ,g$$$$$$$$$$$$$$$P.       ------------
  ,g$$P"     """Y$$.".        OS: Debian GNU/Linux 9.6 (stretch) x86_64
 ,$$P'              `$$$.     Model: VirtualBox 1.2
',$$P       ,ggs.     `$$b:   Kernel: 4.9.0-8-amd64
`d$$'     ,$P"'   .    $$$    Uptime: 2 minutes
 $$P      d$'     ,    $$P    Packages: 1499
 $$:      $$.   -    ,d$$'    Shell: bash 4.4.12
 $$;      Y$b._   _,d$P'      Resolution: 1280x800
 Y$$.    `.`"Y$$$$P"'         DE: LXDE
 `$$b      "-.__              WM: Openbox
  `Y$$                        Theme: Adwaita [GTK3]
   `Y$$.                      Icons: Adwaita [GTK3]
     `$$b.                    Terminal: lxterminal
       `Y$$b.                 CPU: Intel i5-6600K (2) @ 3.5GHz
          `"Y$b._             GPU: VirtualBox Graphics Adapter
              `"""            Memory: 83MB / 2004MB

pi@raspberry:~ $ 
```

No doubt you've heard of the Raspberry Pi? It's hard not to have, as this remarkable, tiny computer has taken the technology world by storm since it was introduced six years ago.

There are several aspects to the Raspberry Pi that make it such a sought-after piece of the computing world. For one, it's cheap, costing around £25 for, what is essentially, a fully working computer. It's small, measuring not much bigger than a credit card. You can employ it to build electronics, using a fully programmable interface and it comes with Raspbian, its own custom-made, Debian-based operating system that includes an office suite alongside many different programming languages and educational resources.

Originally, Raspbian was exclusive to the Pi hardware, as the Raspberry Pi uses an ARM processor to power it. However, the Raspberry Pi Foundation has since released a PC version of Raspbian: Raspberry Pi Desktop.

As with the Pi version, Raspberry Pi Desktop comes with the all the coding, educational and other apps you will ever need. It's quick, stable and works superbly. If you're interested in stretching your Linux experience, then this is definitely one of the top distros to consider.

Equipment You Will Need

The system requirements for successfully installing Linux Mint on to a PC are surprisingly low, so even a computer that's several years old will happily run this distro. However, it's worth checking you have everything in place before proceeding.

MINTY INGREDIENTS

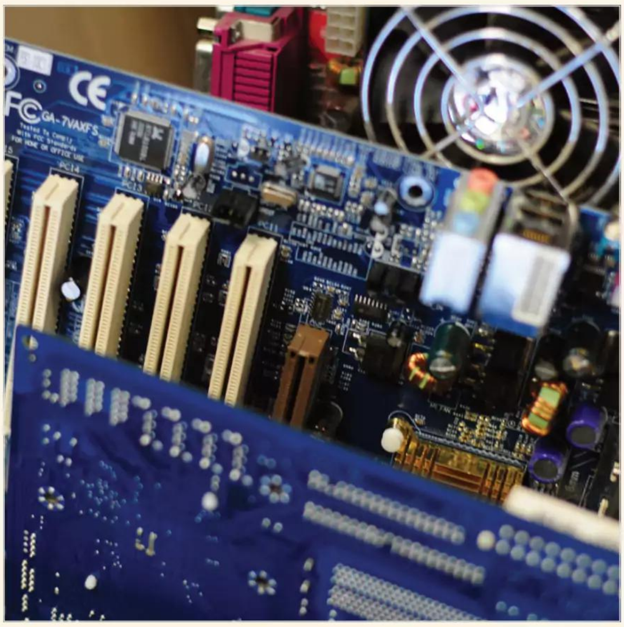
Here's what you'll need to install and run Linux Mint as we work through this book. You have several choices available, so take your time and see which works best for you.

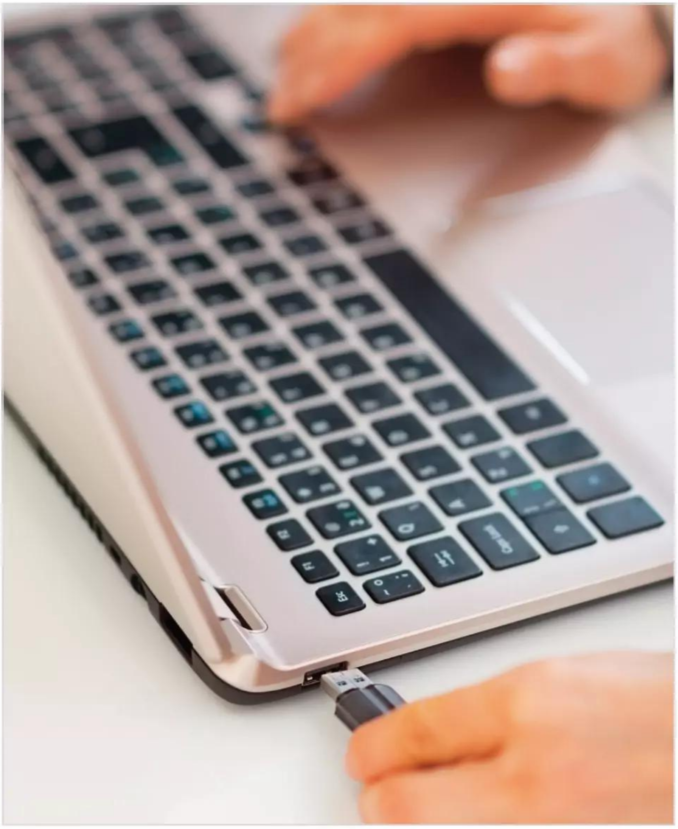
SYSTEM REQUIREMENTS

The minimum system requirements for Linux Mint are as follows:

CPU	700MHz
RAM	512MB
Hard Drive Space	9GB (20GB recommended)
Monitor	1024 x 768 resolution

Obviously the better the system you have, the better and quicker the experience will be.



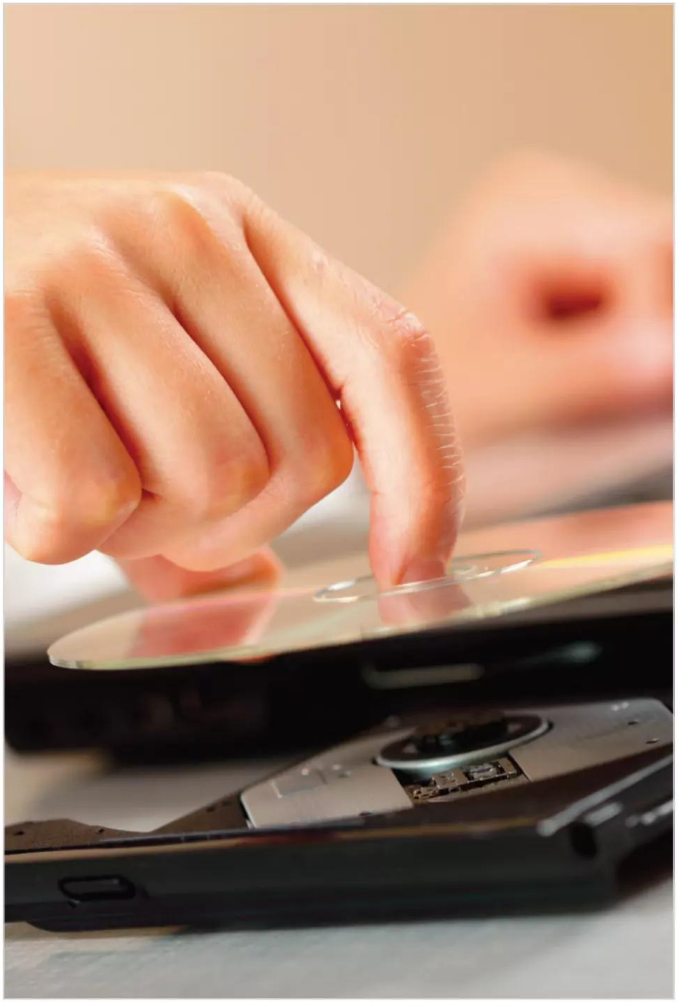


USB INSTALLATION

You can install Linux Mint onto your computer via USB or DVD. We'll look into each a little later on, but if you're already familiar with the process (or you're thinking of USB and just gathering the hardware you'll need), then you're going to need a minimum 4GB USB flash drive/stick to contain the Linux Mint ISO.

DVD INSTALLATION

DVD installation of Linux Mint simply requires a blank DVD-R disc. Of course, you'll also need an optical drive (a DVD Writer drive) before you're able to transfer, or burn, the ISO image to the disc.



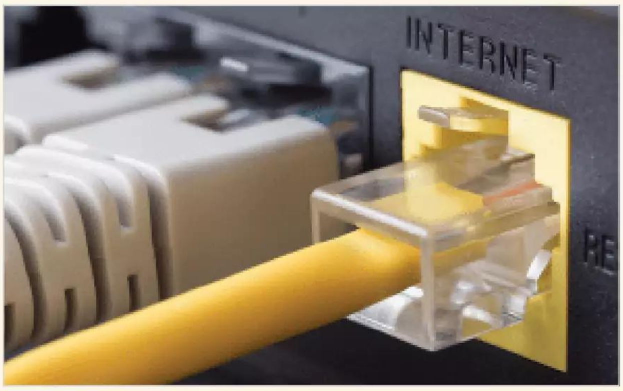
VIRTUAL ENVIRONMENT

Installation to a virtual environment is a favourite method of testing and using Linux distros. Linux Mint works exceedingly well when used in a virtual environment, more on that later. There are many different virtual environment apps available, however VirtualBox, from Oracle, is one of the easiest. You can get the latest version from **www.virtualbox.org**.

VirtualBox

INTERNET CONNECTION

It goes without saying, that an Internet connection is vital for making sure that Linux Mint is resourced with the latest updates and patches, as well as the installation of further software. Although you don't need an Internet connection to use Linux Mint, you'll miss out on a world of free software available for the distro.



MAC HARDWARE

Although Linux Mint can be installed onto a Mac, there's a school of thought that recommends Mac owners use a virtual environment, such as VirtualBox or Parallels; and why not, macOS is already a splendid operating system. If you're looking to breathe new life into an older Mac, make sure it's an Intel CPU model and not the Power-PC models. However, be aware, it's not as pain-free as installing on to a PC.

Finder
File
Edit
View
Go
Window
Help
Black Dog Media Ltd
Wed 11:11

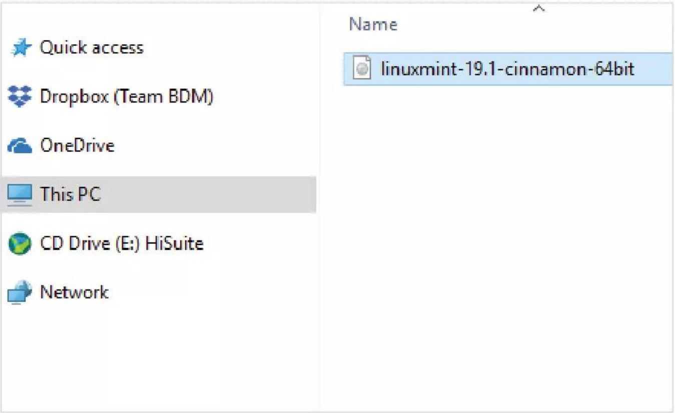
Creating a Linux Installer on Windows

You need to transfer the downloaded Linux ISO to either a DVD or a USB key before you can install it to a computer. This will be a live environment, which will allow you to test the OS prior to installation, but first you need to create the bootable media.

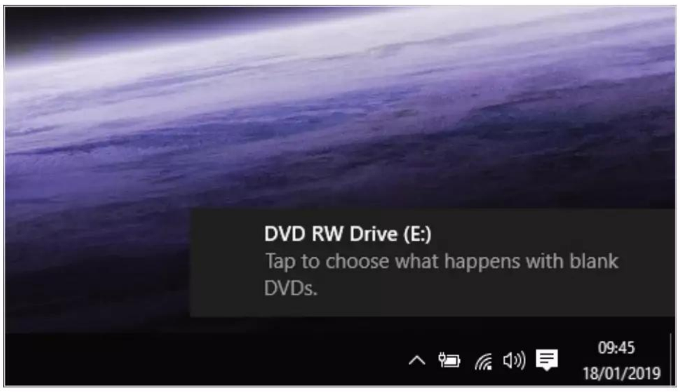
DVD BOOTABLE MEDIA

We're using a Windows 10 PC here to transfer the ISO to a DVD. As long as you're using a version of Windows from 7 onward, the process is extremely easy.

STEP 1 First locate the ISO image of Linux you've already downloaded. In Windows 7, 8.1 and 10 computers, you'll usually find this in the Downloads folder unless, when saving it, you've specified a different location.


Quick access
Dropbox (Team BDM)
OneDrive
This PC
CD Drive (E:) HiSuite
Network
Name
linuxmint-19.1-cinnamon-64bit

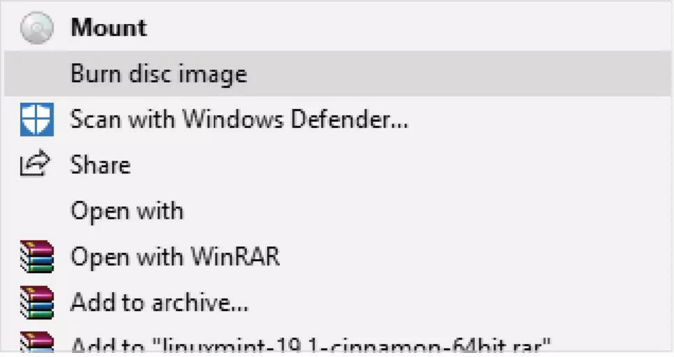
STEP 2 Next, insert a recordable DVD disc into your computer's optical drive. After a few seconds, while the disc is read, Windows will display a pop-up message asking you what to do with the newly inserted disc, ignore this as we're going to use the built-in image burning function.



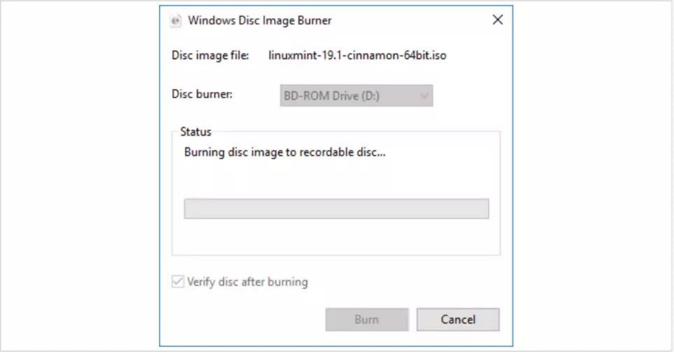
DVD RW Drive (E:)
Tap to choose what happens with blank DVDs.

09:45
18/01/2019

STEP 3 Right-click the Linux ISO and from the menu select Burn Disc Image. Depending on the speed of the PC, it may take a few seconds before anything happens. Don't worry too much, unless it takes more than a minute, in which case it might be worth restarting your PC and trying again. With luck, the Windows Disc Image Burner should launch.


Mount
Burn disc image
Scan with Windows Defender...
Share
Open with
Open with WinRAR
Add to archive...
Add to "linuxmint-19.1-cinnamon-64bit.rar"

STEP 4 With the Windows Disc Image Burner dialogue box open, click on the 'Verify disc after burning' tick box, followed by the **Burn** button. The process should take a few minutes, depending on the speed of your PC's optical drive. Once it's complete, it will run through the verification stage and when done, the optical drive should auto-eject the disc for you.



Windows Disc Image Burner

Disc image file: linuxmint-19.1-cinnamon-64bit.iso

Disc burner: BD-ROM Drive (D:)

Status
Burning disc image to recordable disc...

☑ Verify disc after burning

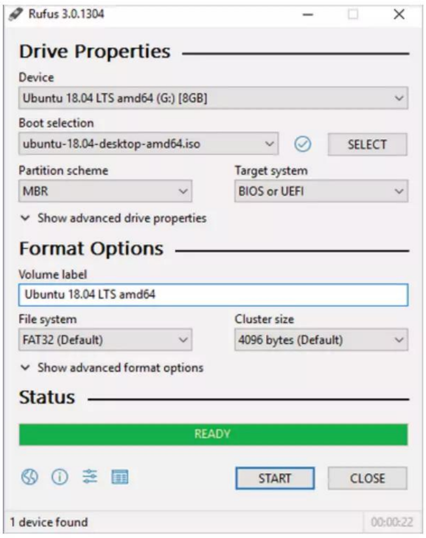
Burn
Cancel

USB BOOTABLE MEDIA

USB media is faster than a DVD and it's often more convenient as most modern PCs don't have an optical drive installed. The process of transferring the image is easy, but you'll need a third-party app first and a USB flash drive of 4GB or more.

STEP 1

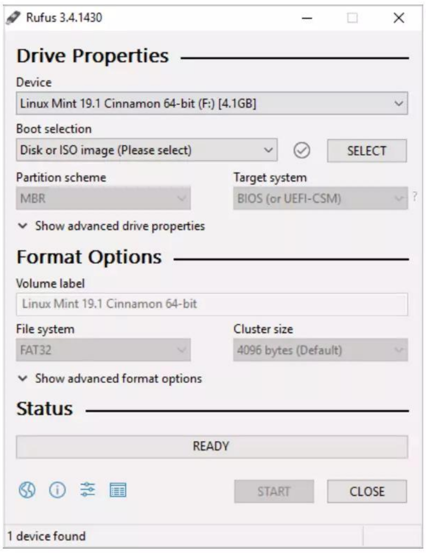
First, open up a web browser and go to **https://rufus.akeo.ie/**. Scroll down the page a little and you'll come to a Download heading, under which you'll see the latest version of Rufus. Left-click on the link to start the download.



Device: Ubuntu 18.04 LTS amd64 (G:) [8GB]
Boot selection: ubuntu-18.04-desktop-amd64.iso
Partition scheme: MBR
Target system: BIOS or UEFI
Show advanced drive properties
Volume label: Ubuntu 18.04 LTS amd64
File system: FAT32 (Default)
Cluster size: 4096 bytes (Default)
Show advanced format options
1 device found
00:00:22

STEP 2

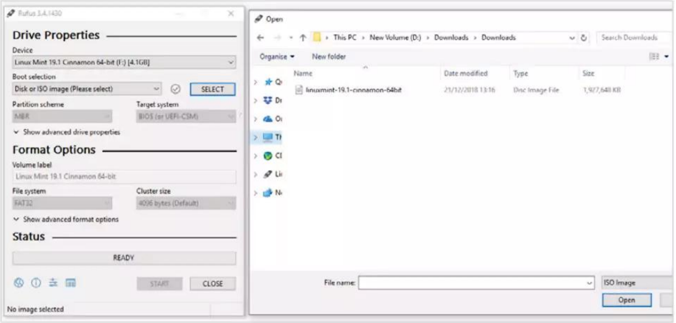
Double-click the downloaded Rufus executable; you can click 'Yes' to the Windows security question and 'Yes' to checking for updates. With Rufus launched, it should have already identified your inserted USB flash drive (if not, just remove and reinsert).



Device: Linux Mint 19.1 Cinnamon 64-bit (F:) [4.1GB]
Boot selection: Disk or ISO image (Please select)
Partition scheme: MBR
Target system: BIOS (or UEFI-CSM)
Show advanced drive properties
Volume label: Linux Mint 19.1 Cinnamon 64-bit
File system: FAT32
Cluster size: 4096 bytes (Default)
Show advanced format options
1 device found

STEP 3

At first glance the Rufus interface can look a little confusing, don't worry though, it's really quite simple. To begin with, click on the SELECT button next to the 'Disk or ISO Image (Please select)' pull-down menu. This will launch a Windows Explorer window from where you can locate and select the Linux ISO.



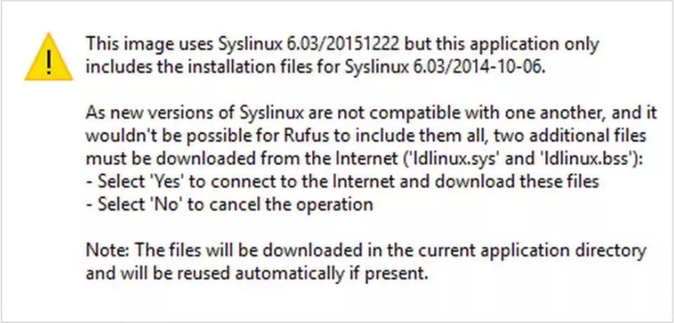
Device: Linux Mint 19.1 Cinnamon 64-bit (F:) [4.1GB]
Boot selection: Disk or ISO image (Please select)
Partition scheme: MBR
Target system: BIOS (or UEFI-CSM)
Show advanced drive properties
Volume label: Linux Mint 19.1 Cinnamon 64-bit
File system: FAT32
Cluster size: 4096 bytes (Default)
Show advanced format options
No image selected

Name	Date modified	Type	Size
linuxmint-19.1-cinnamon-64bit	21/12/2018 13:16	Disc Image File	1,9?7,648 KB

File name: ISO Image

STEP 4

When you're ready, click on the Start button at the bottom of the Rufus app. This may open up another dialogue box asking you to download and use a new version of SysLinux. SysLinux is a selection of boot loaders used to allow a modern PC to access and boot from a USB flash drive. It is necessary, so if asked, click on 'Yes' to continue.



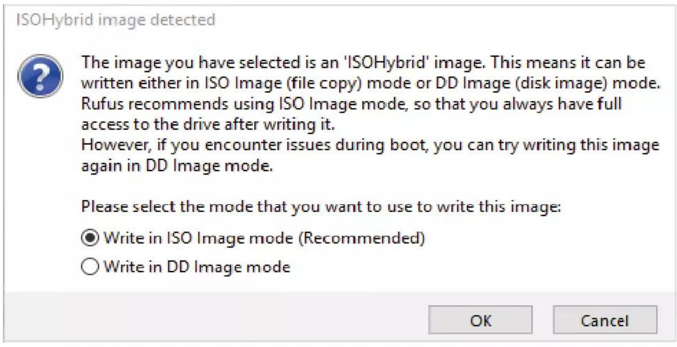
This image uses Syslinux 6.03/20151222 but this application only includes the installation files for Syslinux 6.03/2014-10-06.

As new versions of Syslinux are not compatible with one another, and it wouldn't be possible for Rufus to include them all, two additional files must be downloaded from the Internet ('ldlinux.sys' and 'ldlinux.bss'):
- Select 'Yes' to connect to the Internet and download these files
- Select 'No' to cancel the operation

Note: The files will be downloaded in the current application directory and will be reused automatically if present.

STEP 5

The next step asks in which image mode do you want the Linux ISO to be written to the USB flash drive. Both methods work for different situations, but generally, the recommended ISO Image Mode is the more popular. Make sure this mode is pre-selected and click 'OK' to continue, followed by 'OK' again to confirm the action.



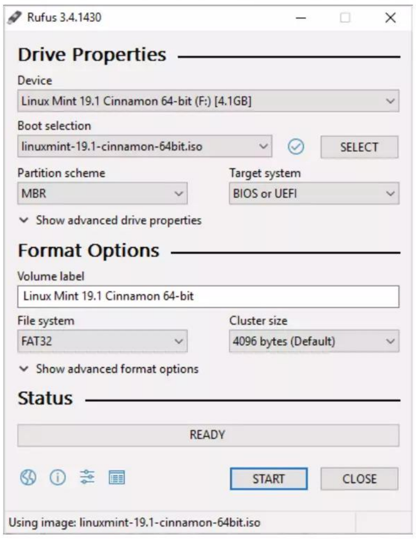
The image you have selected is an 'ISOHybrid' image. This means it can be written either in ISO Image (file copy) mode or DD Image (disk image) mode. Rufus recommends using ISO Image mode, so that you always have full access to the drive after writing it.
However, if you encounter issues during boot, you can try writing this image again in DD Image mode.

Please select the mode that you want to use to write this image:

● Write in ISO Image mode (Recommended)
○ Write in DD Image mode

STEP 6

The Linux ISO is now being transferred to the USB flash drive. The process shouldn't take too long, depending on the speed of the USB device and the PC. During the process, you may find Rufus will auto-open the USB drive in Windows Explorer, don't worry you can minimise or close it if you want. Click on the Close button once the process is complete.



Device: Linux Mint 19.1 Cinnamon 64-bit (F:) [4.1GB]
Boot selection: linuxmint-19.1-cinnamon-64bit.iso
Partition scheme: MBR
Target system: BIOS or UEFI
Show advanced drive properties
Volume label: Linux Mint 19.1 Cinnamon 64-bit
File system: FAT32
Cluster size: 4096 bytes (Default)
Show advanced format options
Using image: linuxmint-19.1-cinnamon-64bit.iso

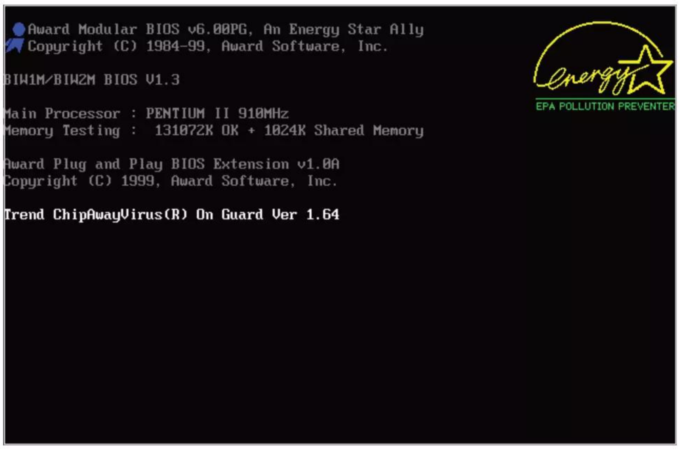
Installing Linux on a PC

Most Linux distros come as a Live Environment. This means you can boot into an actual, fully working distro straight from the DVD or USB you've just created. Let's see how that works and how you go about installing Linux from there.

UEFI BIOS

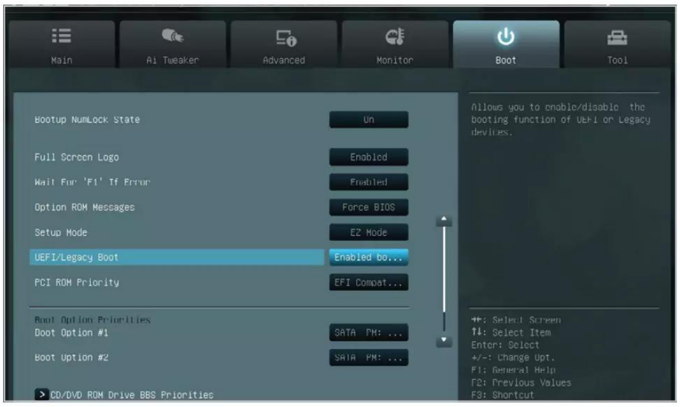
The Unified Extensible Firmware Interface (UEFI) is used to identify hardware and protect a PC during its boot-up process. It replaces the traditional BIOS, but can cause issues when installing Linux.

STEP 1 Insert your DVD or USB flash drive into your PC and, if you haven't already, shutdown Windows. In this instance, we're using the USB boot media but the process is virtually identical. Start the PC and when prompted press the appropriate keys to enter the BIOS or SETUP; these could be, for example, **F2**, **Del** or even **F12**.



```
 Award Modular BIOS v6.00PG, An Energy Star Ally
 Copyright (C) 1984-99, Award Software, Inc.

BIW1M/BIW2M BIOS V1.3

Main Processor : PENTIUM II 910MHz
Memory Testing :  131072K OK + 1024K Shared Memory

Award Plug and Play BIOS Extension v1.0A
Copyright (C) 1999, Award Software, Inc.

Trend ChipAwayVirus(R) On Guard Ver 1.64
```

STEP 2 There are different versions of a UEFI BIOS, so covering them all would be impossible. What you're looking for is a section that details the **Boot Sequence** or **Boot Mode**. Here you'll have the option to turn off UEFI and choose **Legacy**, or disable **Secure Booting**. Most distros work with UEFI but it can be a tricky process to enable it to boot.

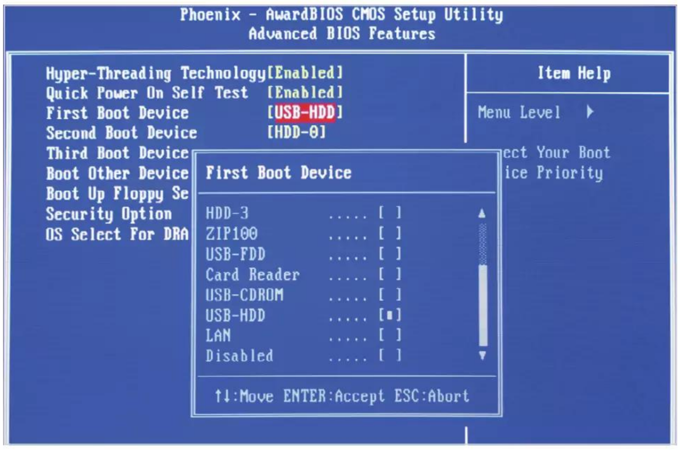


```
Bootup NumLock State            On
Full Screen Logo                Enabled
Wait For 'F1' If Error          Enabled
Option ROM Messages             Force BIOS
Setup Mode                      EZ Mode
UEFI/Legacy Boot                Enabled bo...
PCI ROM Priority                EFI Compat...

Boot Option Priorities
Boot Option #1                  SATA  PM: ...
Boot Option #2                  SATA  PM: ...

> CD/DVD ROM Drive BBS Priorities
```

```
Allows you to enable/disable the
booting function of UEFI or Legacy
devices.
```

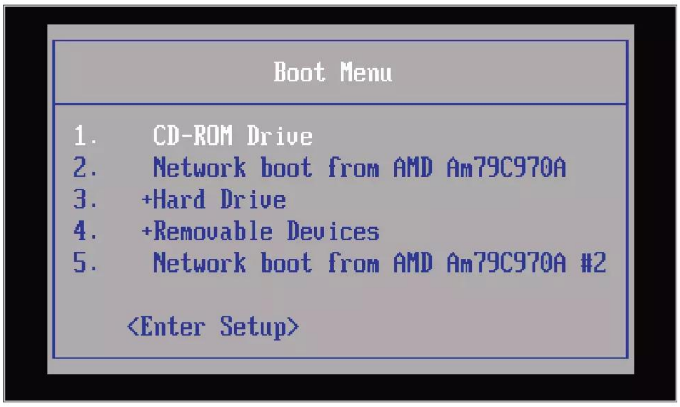
```
→←: Select Screen
↑↓: Select Item
Enter: Select
+/-: Change Opt.
F1: General Help
F2: Previous Values
F3: Shortcut
```

STEP 3 With UEFI turned to Legacy mode, there are now two ways of booting into the Live Environment. The first is via the BIOS you're already in. Locate the **Boot Sequence** and change the first boot device from its original setting, usually Internal HDD or similar, to: **USB Storage Device** for the USB media option; or **DVD Drive**, for the DVD media option.



```
              Phoenix - AwardBIOS CMOS Setup Utility
                      Advanced BIOS Features

  Hyper-Threading Technology[Enabled]          Item Help
  Quick Power On Self Test  [Enabled]
  First Boot Device         [USB-HDD]      Menu Level   ▶
  Second Boot Device        [HDD-0]
  Third Boot Device                        ect Your Boot
  Boot Other Device  First Boot Device     ice Priority
  Boot Up Floppy Se
  Security Option    HDD-3        ..... [ ]
  OS Select For DRA  ZIP100       ..... [ ]
                     USB-FDD      ..... [ ]
                     Card Reader  ..... [ ]
                     USB-CDROM    ..... [ ]
                     USB-HDD      ..... [■]
                     LAN          ..... [ ]
                     Disabled     ..... [ ]

                      ↑↓:Move ENTER:Accept ESC:Abort
```

STEP 4 Alternatively use the Boot Option Menu. With this option, you can press **F12** (or something similar) to display a list of boot media options; from there, you can choose the appropriate boot media. Either way, you can now Save and Exit the BIOS by navigating to the Save & Exit option and choosing **Save Changes and Exit**.

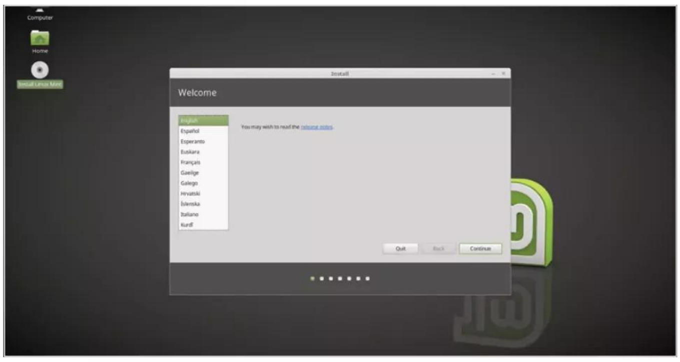


```
                 Boot Menu

 1.   CD-ROM Drive
 2.   Network boot from AMD Am79C970A
 3.  +Hard Drive
 4.  +Removable Devices
 5.   Network boot from AMD Am79C970A #2

     <Enter Setup>
```

INSTALLING LINUX

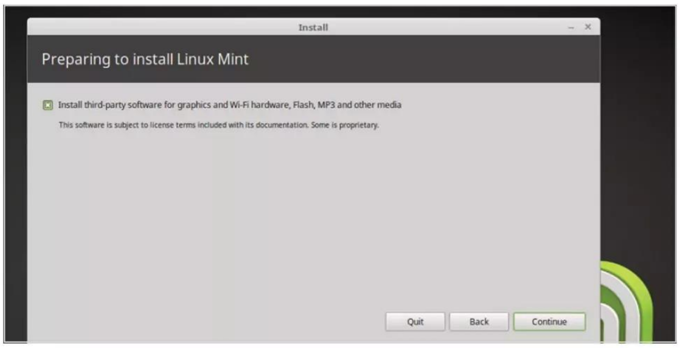
Once the Live Environment has booted, you will see the option to install the distro to your computer. Have a look around and when you're ready, look for the Install option on the desktop.

STEP 1 Providing you're connected to the Internet (if not, then do so now) and you're in the Live Environment, start the installation process by double-clicking on the **Install Linux Mint** icon on the desktop. Other distros will display their own name, of course, but the process is the same. Click **Continue** when you're ready.



You may wish to read the release notes.

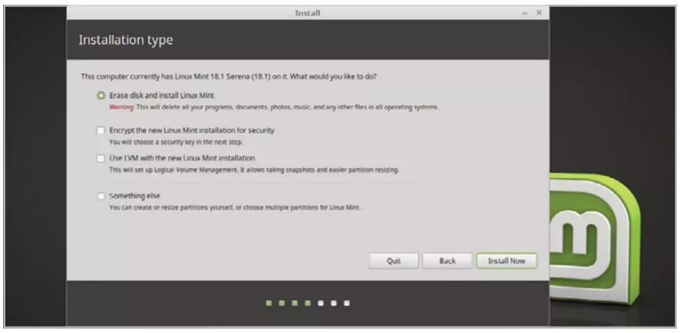
STEP 2 While the installation process is very similar across most Linux distros, some offer different questions during the installation. Generally, the questions aren't too difficult, or technical, but some such as 'Installing third-party software…' can be confusing. In this case, you can click **Continue**, but if you're unsure, have an Internet-connected device available to ask any questions.



☒ Install third-party software for graphics and Wi-Fi hardware, Flash, MP3 and other media

This software is subject to license terms included with its documentation. Some is proprietary.

STEP 3 When installing a new operating system it's recommended that you wipe the old OS, replacing it with the new. When you reach this stage of the installation process, ensure the 'Erase disk and install Linux…' option is selected. NOTE: **This will completely wipe Windows 10 from your computer; make sure you have backups of all your personal files and data.**



This computer currently has Linux Mint 18.1 Serena (18.1) on it. What would you like to do?

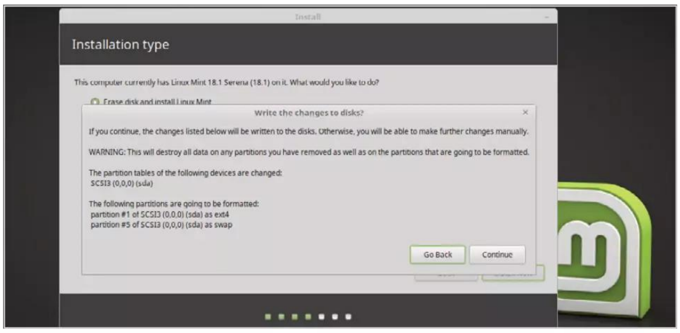
● Erase disk and install Linux Mint
Warning: This will delete all your programs, documents, photos, music, and any other files in all operating systems.

☐ Encrypt the new Linux Mint installation for security
You will choose a security key in the next step.

☐ Use LVM with the new Linux Mint installation
This will set up Logical Volume Management. It allows taking snapshots and easier partition resizing.

○ Something else
You can create or resize partitions yourself, or choose multiple partitions for Linux Mint.

STEP 4 Before the installation process begins, you're asked if the choice you made regarding the erasure of the hard drive is correct. This is your last chance to back out. If you're certain you don't mind wiping everything and starting again with Linux Mint, click **Continue**. If you need to back up your files remove the Linux disc/USB, reboot, back up and start again.



This computer currently has Linux Mint 18.1 Serena (18.1) on it. What would you like to do?

● Erase disk and install Linux Mint

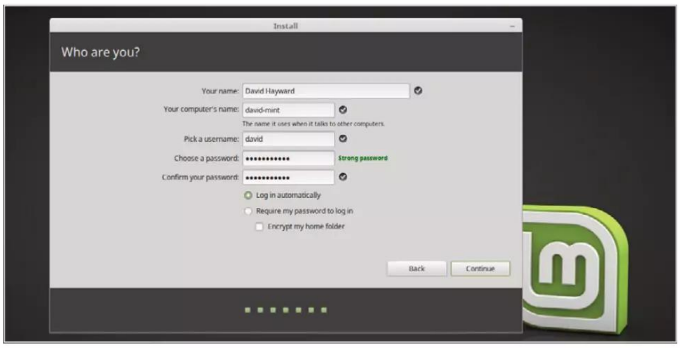
If you continue, the changes listed below will be written to the disks. Otherwise, you will be able to make further changes manually.

WARNING: This will destroy all data on any partitions you have removed as well as on the partitions that are going to be formatted.

The partition tables of the following devices are changed:
SCSI3 (0,0,0) (sda)

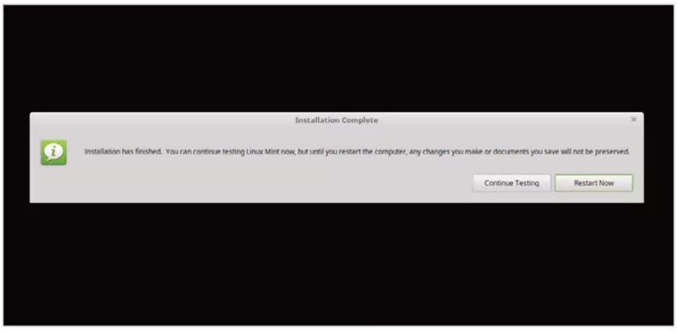
The following partitions are going to be formatted:
partition #1 of SCSI3 (0,0,0) (sda) as ext4
partition #5 of SCSI3 (0,0,0) (sda) as swap

STEP 5 Eventually you will be asked to set up your Linux username and password. Enter your Name to begin with, then **Computer Name** – which is the name used to identify it on the network. Next, choose a **Username**, followed by a good Password. You can tick the **Login Automatically** option, but leave the Encrypt Home Folder option for now.



Your name: David Hayward
Your computer's name: david-mint
The name it uses when it talks to other computers.
Pick a username: david
Choose a password: ••••••••••• Strong password
Confirm your password: •••••••••••
● Log in automatically
○ Require my password to log in
☐ Encrypt my home folder

STEP 6 The installation process can be quick and there may be more questions to answer, or it may simply start installing Linux based on your previous answers. Either way, you will end up being asked to Continue Testing the Live Environment, or Restarting to use the newly installed OS. If you're ready to use Linux, then click **Restart Now**.



Installation has finished. You can continue testing Linux Mint now, but until you restart the computer, any changes you make or documents you save will not be preserved.

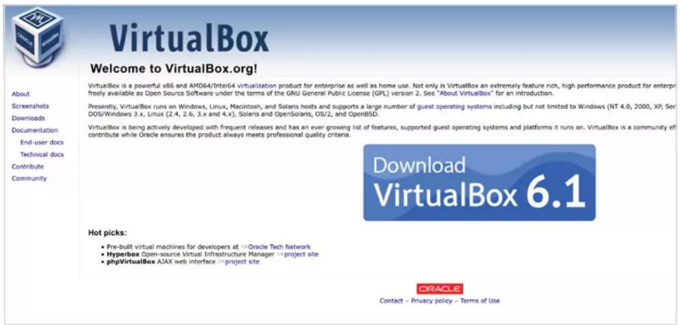
Installing a Virtual Environment

A Virtual Environment is a simulated computer system. Using a Virtual Machine, you can mimic a standard PC and install an entire operating system on it without affecting the one installed on your computer. It's a great way to test and use Linux, while still having Windows 10 as your main OS.

GOING VIRTUAL

Using a Virtual Machine (VM) will take resources from your computer: memory, hard drive space, processor usage and so on. So make sure you've got enough of each before commencing.

STEP 1 We're using VirtualBox in this instance, as it's one of the easiest virtual environments to get to grips with. Enter **www.virtualbox.org** and click on **'Download VirtualBox'**. This will take you to the main download page. Locate the correct host for your system: Windows or Mac – the Host is the current installed operating system and click the link to begin the download.


VirtualBox

Welcome to VirtualBox.org!

VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterpr freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2. See "About VirtualBox" for an introduction.

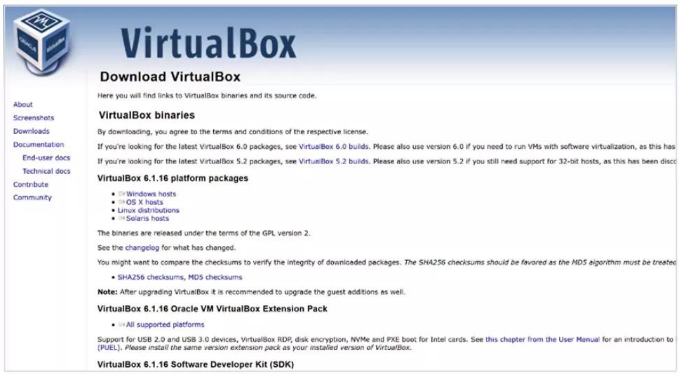
Presently, VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts and supports a large number of guest operating systems including but not limited to Windows (NT 4.0, 2000, XP, Ser DOS/Windows 3.x, Linux (2.4, 2.6, 3.x and 4.x), Solaris and OpenSolaris, OS/2, and OpenBSD.

VirtualBox is being actively developed with frequent releases and has an ever growing list of features, supported guest operating systems and platforms it runs on. VirtualBox is a community ef contribute while Oracle ensures the product always meets professional quality criteria.

Hot picks:

- Pre-built virtual machines for developers at Oracle Tech Network
- **Hyperbox** Open-source Virtual Infrastructure Manager project site
- **phpVirtualBox** AJAX web interface project site

STEP 2 Next, while still at the VirtualBox download page, locate the VirtualBox Extension Pack link. The Extension Pack supports USB devices, as well as numerous other extras that can help make the VM environment a more accurate emulation of a 'real' computer.


VirtualBox

Download VirtualBox

Here you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

By downloading, you agree to the terms and conditions of the respective license.

If you're looking for the latest VirtualBox 6.0 packages, see VirtualBox 6.0 builds. Please also use version 6.0 if you need to run VMs with software virtualization, as this has

If you're looking for the latest VirtualBox 5.2 packages, see VirtualBox 5.2 builds. Please also use version 5.2 if you still need support for 32-bit hosts, as this has been disc

VirtualBox 6.1.16 platform packages

- Windows hosts
- OS X hosts
- Linux distributions
- Solaris hosts

The binaries are released under the terms of the GPL version 2.

See the changelog for what has changed.

You might want to compare the checksums to verify the integrity of downloaded packages. *The SHA256 checksums should be favored as the MD5 algorithm must be treated*

- SHA256 checksums, MD5 checksums

Note: After upgrading VirtualBox it is recommended to upgrade the guest additions as well.

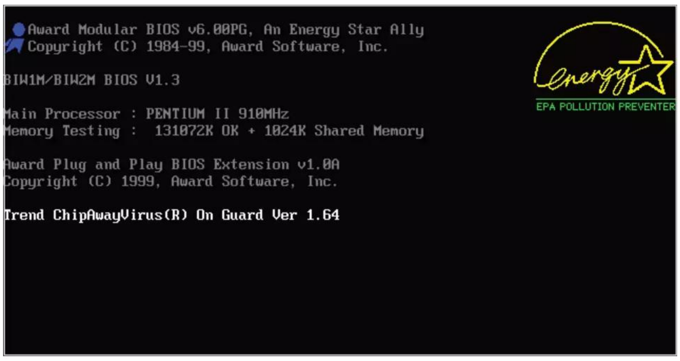
VirtualBox 6.1.16 Oracle VM VirtualBox Extension Pack

- All supported platforms

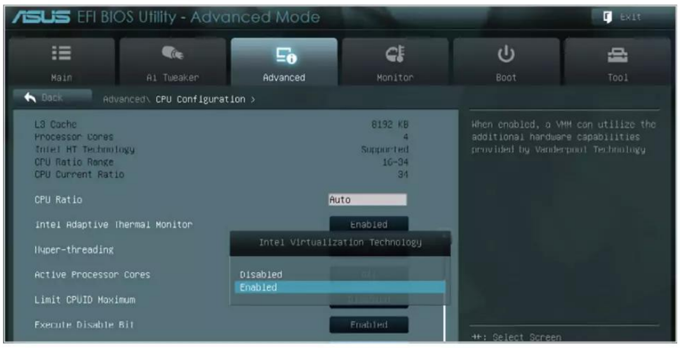
Support for USB 2.0 and USB 3.0 devices, VirtualBox RDP, disk encryption, NVMe and PXE boot for Intel cards. See this chapter from the User Manual for an introduction to (PUEL). *Please install the same version extension pack as your installed version of VirtualBox.*

VirtualBox 6.1.16 Software Developer Kit (SDK)

STEP 3 With the correct packages downloaded, before we install anything, we need to make sure that the computer you're using is able to host a VM. To do this, reboot the computer and enter the BIOS. As the computer starts up, press the **Del**, **F2**, or whichever key is necessary to **Enter Setup**.


```
Award Modular BIOS v6.00PG, An Energy Star Ally
Copyright (C) 1984-99, Award Software, Inc.

BIW1M/BIW2M BIOS V1.3

Main Processor : PENTIUM II 910MHz
Memory Testing :  131072K OK + 1024K Shared Memory

Award Plug and Play BIOS Extension v1.0A
Copyright (C) 1999, Award Software, Inc.

Trend ChipAwayVirus(R) On Guard Ver 1.64
```

STEP 4 As each BIOS is laid out differently, it's very difficult to assess where to look in each personal example. However, as a general rule of thumb, you're looking for Intel Virtualisation Technology, or simply Virtualisation, usually within the Advanced section of the BIOS. When you've located it, Enable it, save the settings, exit the BIOS and reboot the computer.

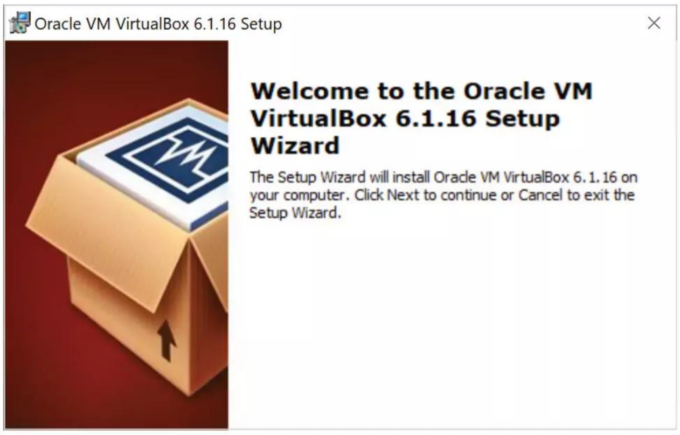

ASUS EFI BIOS Utility - Advanced Mode
Exit
Main
Ai Tweaker
Advanced
Monitor
Boot
Tool
Back
Advanced\ CPU Configuration >

```
L3 Cache                                  8192 KB
Processor Cores                                 4
Intel HT Technology                     Supported
CPU Ratio Range                             16-34
CPU Current Ratio                              34

CPU Ratio                                 Auto

Intel Adaptive Thermal Monitor            Enabled

Hyper-threading

Active Processor Cores

Limit CPUID Maximum

Execute Disable Bit                       Enabled
```

```
Intel Virtualization Technology
Disabled
Enabled
```

When enabled, a VMM can utilize the additional hardware capabilities provided by Vanderpool Technology

STEP 5 With the computer back up and running, locate the downloaded main VirtualBox application and double-click to begin the installation process. Click Next to continue, when you're ready.

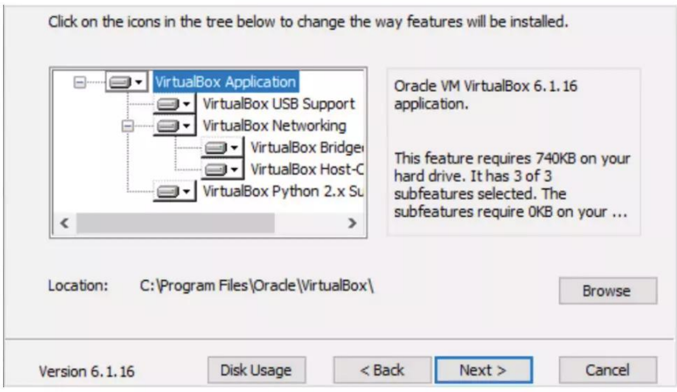


Oracle VM VirtualBox 6.1.16 Setup

Welcome to the Oracle VM VirtualBox 6.1.16 Setup Wizard

The Setup Wizard will install Oracle VM VirtualBox 6.1.16 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

STEP 6 The default installation location of VirtualBox should satisfy most users, but if you have any special location requirements, click on the 'Browse' button and change the install folder. Also, make sure that all the icons in the VirtualBox feature tree are selected and none of them have a red X next to them. Click Next to move on.



Click on the icons in the tree below to change the way features will be installed.

- VirtualBox Application
  - VirtualBox USB Support
  - VirtualBox Networking
    - VirtualBox Bridge
    - VirtualBox Host-C
  - VirtualBox Python 2.x Su

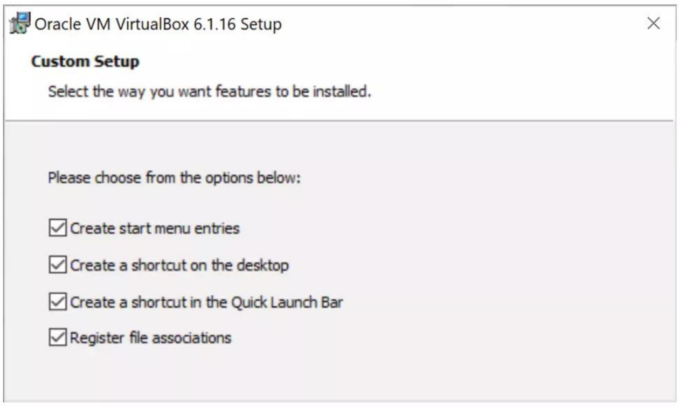
Oracle VM VirtualBox 6.1.16 application.

This feature requires 740KB on your hard drive. It has 3 of 3 subfeatures selected. The subfeatures require 0KB on your ...

Location: C:\Program Files\Oracle\VirtualBox\

Version 6.1.16

STEP 7 This section can be left to the defaults, should you wish. It simply makes life a little easier when dealing with VMs, especially when dealing with downloaded VMs, as you may encounter in the future. Again, clicking Next will move you on to the next stage.



Oracle VM VirtualBox 6.1.16 Setup

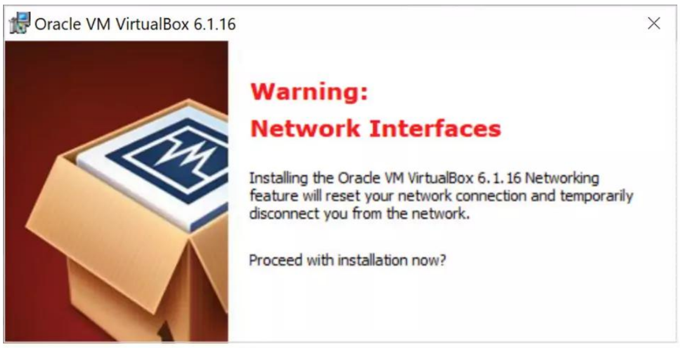
Custom Setup

Select the way you want features to be installed.

Please choose from the options below:

☑ Create start menu entries
☑ Create a shortcut on the desktop
☑ Create a shortcut in the Quick Launch Bar
☑ Register file associations

STEP 8 When installing VirtualBox, your network connection will be disabled for a very brief period. This is due to VirtualBox creating a linked, virtual network connection so that any VM installed will be able to access the Internet and your home network resources, via the computer's, already established, network connection. Click Yes, then, Install to begin the installation.



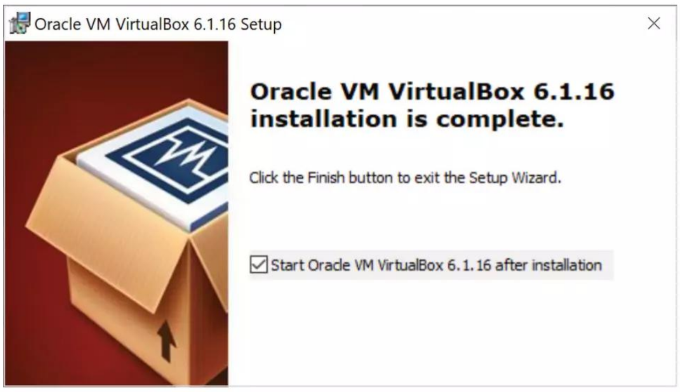
Oracle VM VirtualBox 6.1.16

Warning:
Network Interfaces

Installing the Oracle VM VirtualBox 6.1.16 Networking feature will reset your network connection and temporarily disconnect you from the network.

Proceed with installation now?

STEP 9 You'll probably be asked by Windows to accept a security notification, click Yes for this and next you may encounter a dialogue box asking you to trust the installation from Oracle, again, click yes and accept the installation of the VirtualBox application. When it's complete, click finish to start VirtualBox.



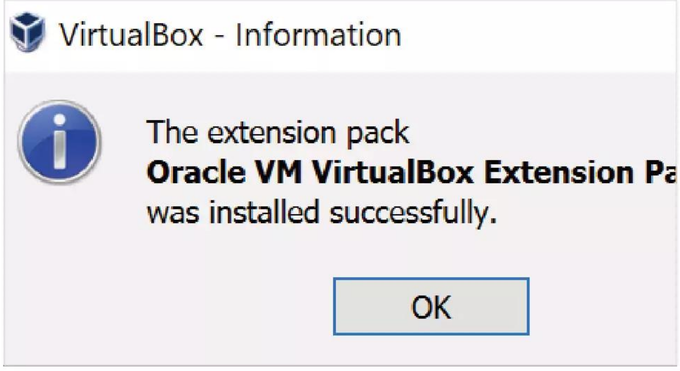
Oracle VM VirtualBox 6.1.16 Setup

Oracle VM VirtualBox 6.1.16 installation is complete.

Click the Finish button to exit the Setup Wizard.

☑ Start Oracle VM VirtualBox 6.1.16 after installation

STEP 10 With VirtualBox up and running, you can now install the VirtualBox Extension Pack. Locate the downloaded add-on and double-click. There may be a short pause while VirtualBox analyses the pack, but you'll eventually receive a message to install it; click Install to begin the process, scroll down the next screen to accept the agreement and click 'I Agree'.



VirtualBox - Information

The extension pack
Oracle VM VirtualBox Extension Pa
was installed successfully.

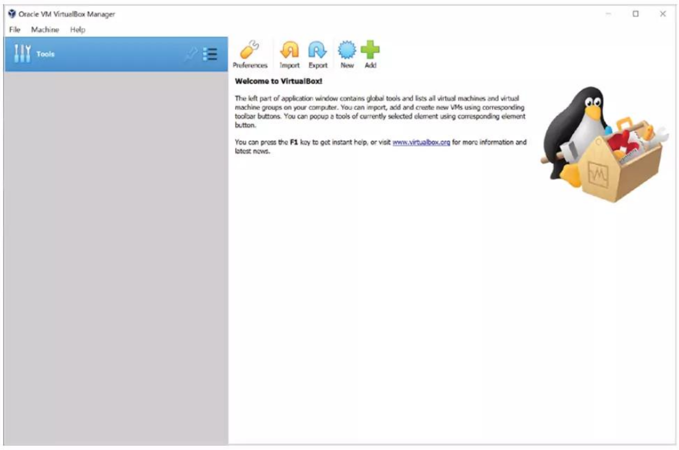
Installing Linux in a Virtual Environment

With Oracle's VirtualBox now up and running, the next task is to create the Virtual Machine (VM) environment into which you will install Linux. This process won't affect your currently installed operating system, which is why a VM is a great choice.

CREATING THE VM

There are plenty of options to choose from when creating a VM. For now though, we'll setup a VM adequate to run the excellent Linux Mint, and perform well.

STEP 1 With VirtualBox open, click on the **New** icon in the top-middle of the right-hand panel of the app. This will open the new VM Wizard.

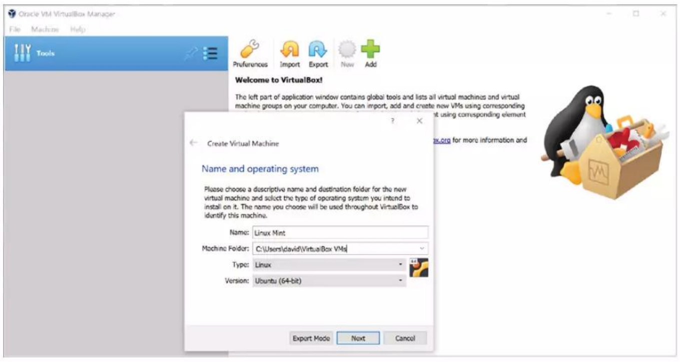

Oracle VM VirtualBox Manager
File
Machine
Help
Tools
Preferences
Import
Export
New
Add

Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

STEP 2 In the box next to Name type **Linux Mint**, and VirtualBox should automatically choose Linux as the Type and Ubuntu (64-bit) as the Version, if not then use the drop-down boxes to select the correct settings (remember Mint mainstream is based on Ubuntu). Click **Next** when you're ready to proceed.

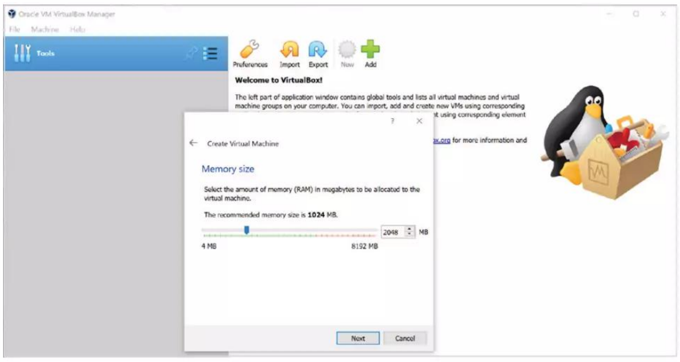

Oracle VM VirtualBox Manager
File
Machine
Help
Tools
Preferences
Import
Export
New
Add
Welcome to VirtualBox!
Create Virtual Machine
Export Mode
Next
Cancel

Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: Linux Mint
Machine Folder: C:\Users\david\VirtualBox VMs
Type: Linux
Version: Ubuntu (64-bit)

STEP 3 The next section will define the amount of system memory, or RAM, the VM has allocated. Remember this amount will be taken from the available memory installed in your computer, so don't give the VM too much. For example, we have 8GB of memory installed and we're giving 2GB (2048MB) to the VM. When you're ready, click **Next** to continue.


Oracle VM VirtualBox Manager
File
Machine
Help
Tools
Preferences
Import
Export
New
Add
Welcome to VirtualBox!
Create Virtual Machine
4 MB
8192 MB
MB
Next
Cancel

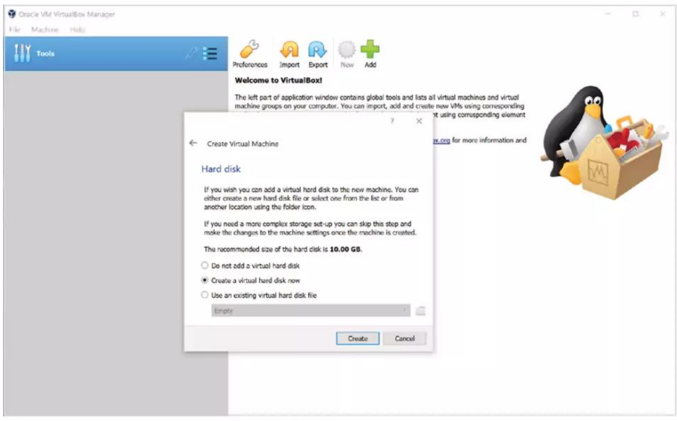
Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024** MB.

2048 MB

STEP 4 This section is where you'll start to create the virtual hard disk that the VM will use to install Mint on to. The default option, 'Create a virtual hard disk now', is the one we're using. Click **Create** to move on.


Oracle VM VirtualBox Manager
File
Machine
Help
Tools
Preferences
Import
Export
New
Add
Welcome to VirtualBox!
Create Virtual Machine
Empty
Create
Cancel

Hard disk

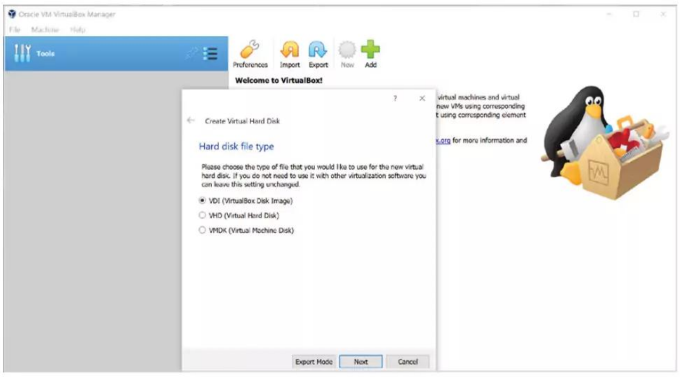
If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **10.00 GB**.

○ Do not add a virtual hard disk
● Create a virtual hard disk now
○ Use an existing virtual hard disk file

STEP 5 The pop-up window that appears after clicking Create is asking you what type of virtual hard disk you want to create. We're going to use the default VDI (VirtualBox Disk Image) in this case, as the others are often used to move VMs from one VM application to the next. Make sure VDI is selected, and click **Next**.


Oracle VM VirtualBox Manager
File Machine Help
Tools
Preferences
Import
Export
New
Add
Welcome to VirtualBox!

Create Virtual Hard Disk

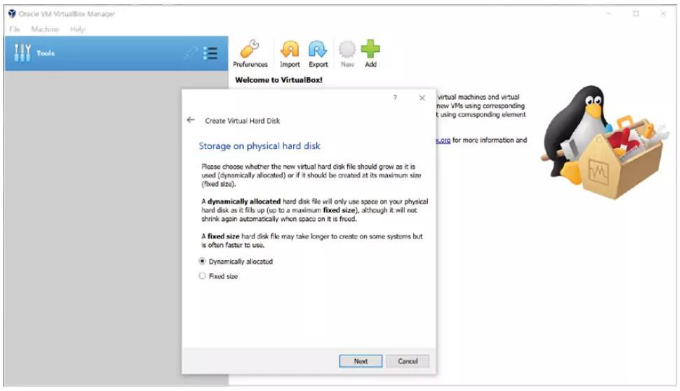
Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

● VDI (VirtualBox Disk Image)
○ VHD (Virtual Hard Disk)
○ VMDK (Virtual Machine Disk)

Export Mode
Next
Cancel

STEP 6 The question of whether to opt for Dynamically or Fixed sized virtual hard disks may come across as being somewhat confusing to the newcomer. Basically, a Dynamically Allocated virtual hard disk is a more flexible storage management option. It won't take up much space within your physical hard disk to begin with either. Ensure **Dynamically Allocated** is selected, and click **Next**.


Oracle VM VirtualBox Manager
File Machine Help
Tools
Preferences
Import
Export
New
Add
Welcome to VirtualBox!

Create Virtual Hard Disk

Storage on physical hard disk

Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

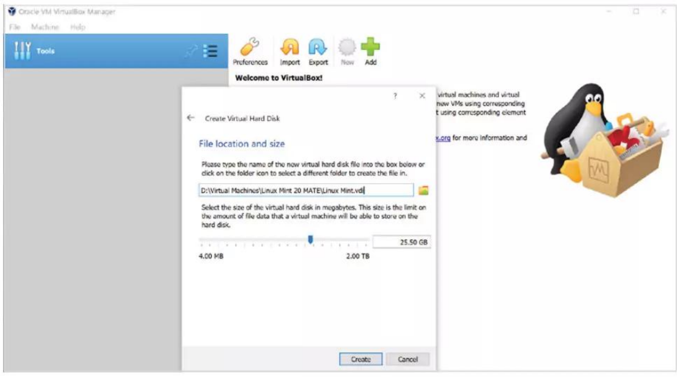
A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

● Dynamically allocated
○ Fixed size

Next
Cancel

STEP 7 The virtual hard disk will be a single folder, up to the size you state in this section. Ensure the location of the virtual hard disk, on your computer, has enough free space available. For example, we've used a bigger storage option on our D:\ drive, named it Linux Mint, and allocated 25.50GB of space to the virtual hard disk.


Oracle VM VirtualBox Manager
File Machine Help
Tools
Preferences
Import
Export
New
Add
Welcome to VirtualBox!

Create Virtual Hard Disk

File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

D:\Virtual Machines\Linux Mint 20 MATE\Linux Mint.vdi

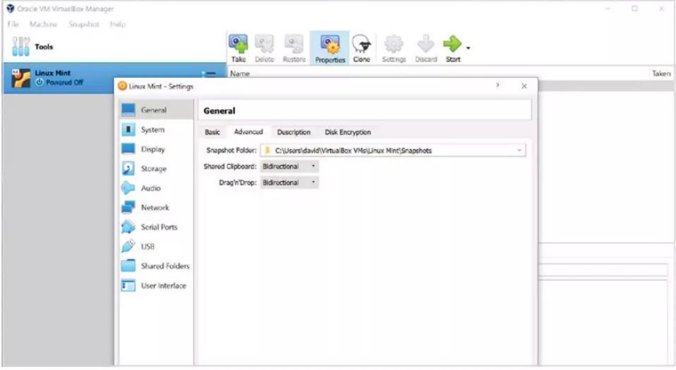
Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

25.50 GB

4.00 MB 2.00 TB

Create
Cancel

STEP 8 After clicking Create the initial setup of the VM is complete; you should now be looking at the newly created VM within the VirtualBox application. Before you begin though, click the **Settings** button from the top of the right-hand panel, and within the General section click the **Advanced** tab. Using the pull-down menus, choose 'Bidirectional' for both Shared Clipboard and Drag'n'Drop.


Oracle VM VirtualBox Manager
File Machine Snapshot Help
Tools
Linux Mint
Powered Off
Take
Delete
Restore
Properties
Clone
Settings
Discard
Start
Name
Taken
General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface

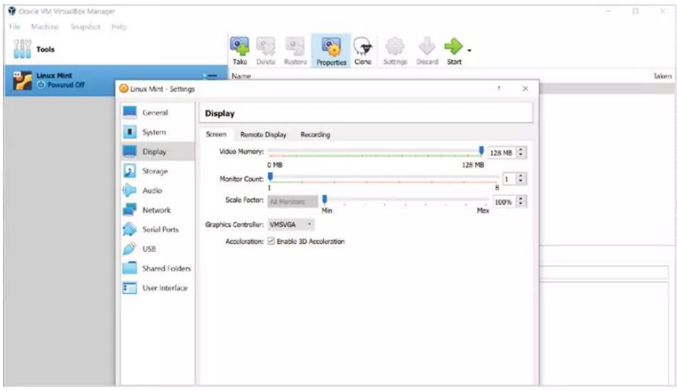
Linux Mint - Settings

General

Basic | Advanced | Description | Disk Encryption

Snapshot Folder: C:\Users\david\VirtualBox VMs\Linux Mint\Snapshots
Shared Clipboard: Bidirectional
Drag'n'Drop: Bidirectional

STEP 9 Follow that by clicking on the **System** section, then the **Processor** tab. Depending on your CPU allocate as many cores as you can without detriment to your host system; we've opted for two CPUs. Now click on the **Display** section, slide the **Video Memory** up to the maximum, and tick 'Enable 3D Acceleration'. Click **OK** to commit the new settings.


Oracle VM VirtualBox Manager
File Machine Snapshot Help
Tools
Linux Mint
Powered Off
Take
Delete
Restore
Properties
Clone
Settings
Discard
Start
Name
Taken
General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface

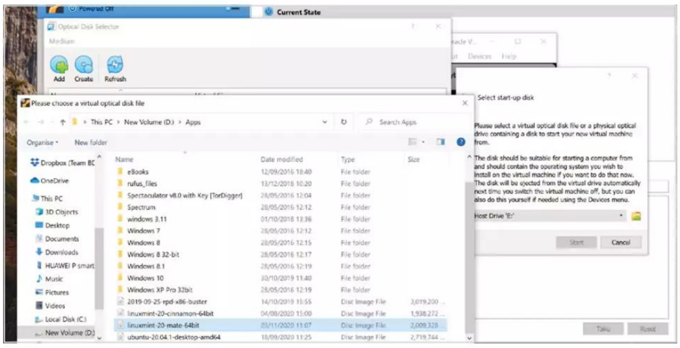
Linux Mint - Settings

Display

Screen | Remote Display | Recording

Video Memory: 128 MB (0 MB – 128 MB)
Monitor Count: 1 (1 – 8)
Scale Factor: All Monitors 100% (Min – Max)
Graphics Controller: VMSVGA
Acceleration: ☑ Enable 3D Acceleration

STEP 10 Click on the Start button and use the explorer button in the 'Select Start-up Disk' window; the explorer button is a folder with a green arrow. Click Add in the new pop-up window, to locate the downloaded ISO of Mint; and click Open to select the ISO. Now click the Start button to boot the VM with the Linux Mint Live Environment. You can now install Linux as per the standard PC installation requirements.


Powered Off
Current State
Optical Disk Selector
Medium
Add
Create
Refresh
Organise
New folder
Dropbox (Team BC
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
HUAWEI P smart
Music
Pictures
Videos
Local Disk (C:)
New Volume (D:)
Devices
Help
Take
Reset

Please choose a virtual optical disk file

This PC > New Volume (D:) > Apps

Name	Date modified	Type	Size
eBooks	12/09/2016 18:40	File folder	
rufus_files	13/12/2018 10:20	File folder	
Spectaculator v8.0 with Key [TorDigger]	28/05/2016 12:04	File folder	
Spectrum	28/05/2016 12:12	File folder	
windows 3.11	01/10/2018 13:36	File folder	
Windows 7	28/05/2016 12:12	File folder	
Windows 8	28/05/2016 12:15	File folder	
Windows 8 32-bit	28/05/2016 12:17	File folder	
Windows 8.1	28/05/2016 12:19	File folder	
Windows 10	30/10/2019 11:40	File folder	
Windows XP Pro 32bit	28/05/2016 12:19	File folder	
2019-09-25-rpd-x86-buster	14/10/2019 11:55	Disc Image File	3,019,200 ...
linuxmint-20-cinnamon-64bit	04/08/2020 15:00	Disc Image File	1,938,272 ...
linuxmint-20-mate-64bit	03/11/2020 11:07	Disc Image File	2,009,328 ...
ubuntu-20.04.1-desktop-amd64	18/09/2020 11:25	Disc Image File	2,719,744 ...

Select start-up disk

Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

Host Drive 'E:'

Start
Cancel

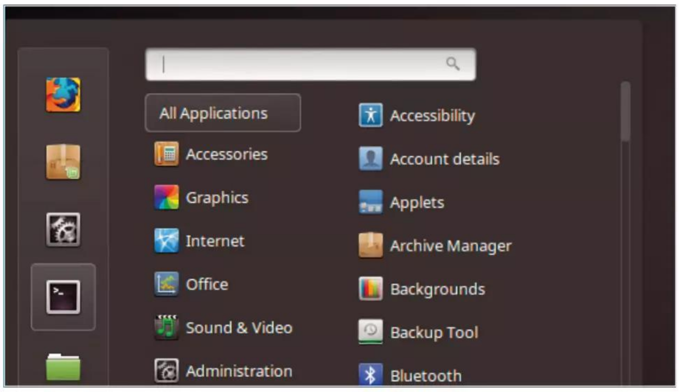
Getting Ready to Code in Linux

Coding in Linux mostly happens in the Terminal or the Command Line. While it can be a scary looking place to begin with, the Terminal is an extremely powerful environment. Before you can start to code, it's best to master the Terminal.

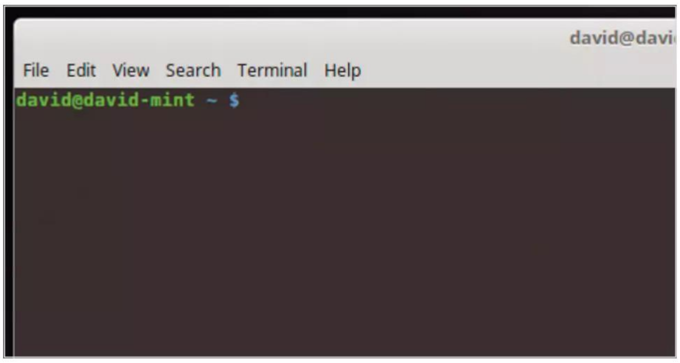
TAKING COMMAND

The command line is at the core of Linux and when you program with it, this is called scripting. These are self-contained programs designed to be run in the Terminal.

STEP 1 The Terminal is where you begin your journey with Linux, through the command line and thus any scripting from. In Linux Mint, it can be accessed by clicking on the Menu followed by the Terminal icon in the panel, or entering 'Terminal' into the search bar.

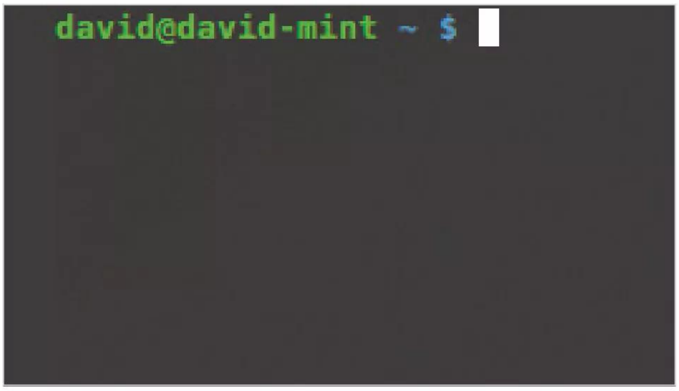

All Applications
Accessories
Graphics
Internet
Office
Sound & Video
Administration
Accessibility
Account details
Applets
Archive Manager
Backgrounds
Backup Tool
Bluetooth

STEP 2 The Terminal will give you access to the Linux Mint Shell, called BASH; this gives you access to the underlying operating system, which is why scripting is such a powerful language to learn and use. Everything in Mint, and Linux as a whole, including the desktop and GUI, is a module running from the command line.


david@davi
File Edit View Search Terminal Help

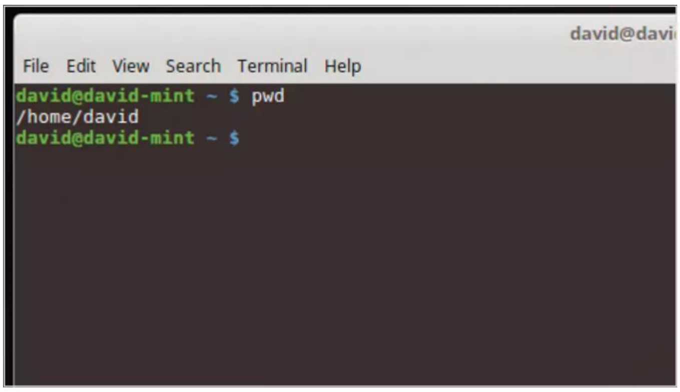
```
david@david-mint ~ $
```

STEP 3 What you currently see in the Terminal is your login name followed by the name of the computer, as you named it when you first installed the OS on to the computer. The line then ends with the current folder name; at first this is just a tilde (~), which means your Home folder.



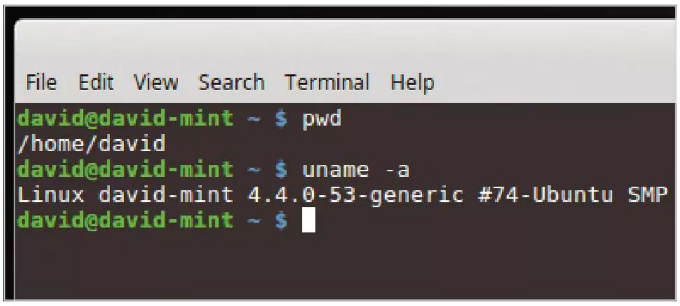
```
david@david-mint ~ $
```

STEP 4 The flashing cursor at the very end of the line is where your text-based commands will be entered. You can begin to experiment with a simple command, Print Working Directory (pwd), which will output to the screen the current folder you're in. Type: `pwd` and press Enter.


david@davi
File Edit View Search Terminal Help

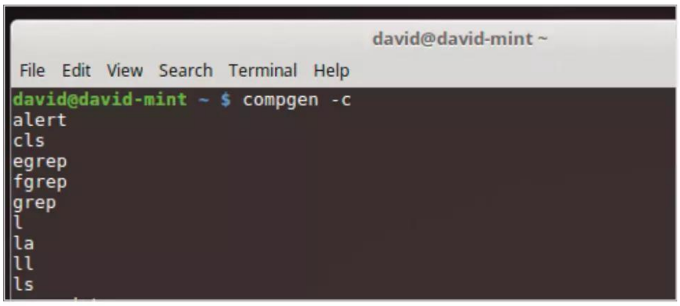
```
david@david-mint ~ $ pwd
/home/david
david@david-mint ~ $
```

STEP 5 All the commands you enter will work in the same manner. You enter the command, include any parameters to extend the use of the command and press Enter to execute the command line you've entered. Now type: `uname -a` and press Enter. This will display information regarding Linux Mint. In scripting, you can use all the Linux command-line commands within your own scripts.


File Edit View Search Terminal Help

```
david@david-mint ~ $ pwd
/home/david
david@david-mint ~ $ uname -a
Linux david-mint 4.4.0-53-generic #74-Ubuntu SMP
david@david-mint ~ $ 
```

STEP 6 The list of available Linux commands is vast, with some simply returning the current working directory, while others are capable of deleting the entire system in an instant. Getting to know the commands is part of learning how to script. By using the wrong command, you could end up wiping your computer. Type `compgen -c` to view the available commands.

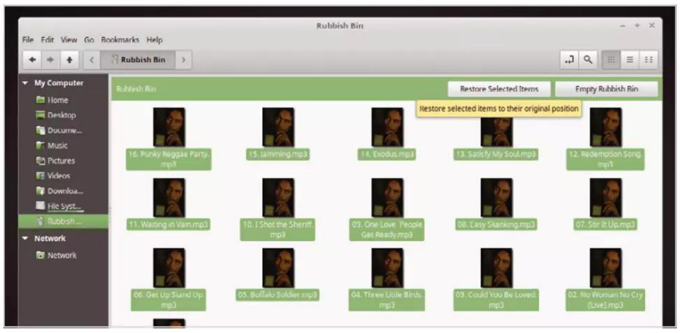

david@david-mint ~
File Edit View Search Terminal Help

```
david@david-mint ~ $ compgen -c
alert
cls
egrep
fgrep
grep
l
la
ll
ls
```

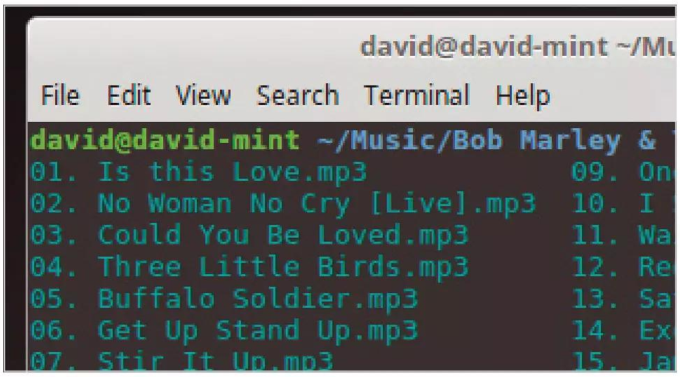
HERE BE DRAGONS!

There's an urban myth on the Internet that an employee at Disney Pixar nearly ruined the animated movie Toy Story by inadvertently entering the wrong Linux command and deleting the entire system the film was stored on.

STEP 1 Having access to the Terminal means you're bypassing the GUI desktop method of working with the system. The Terminal is a far more powerful environment than the desktop, which has several safeguards in place in case you accidentally delete all your work, such as Rubbish Bin to recover deleted files.

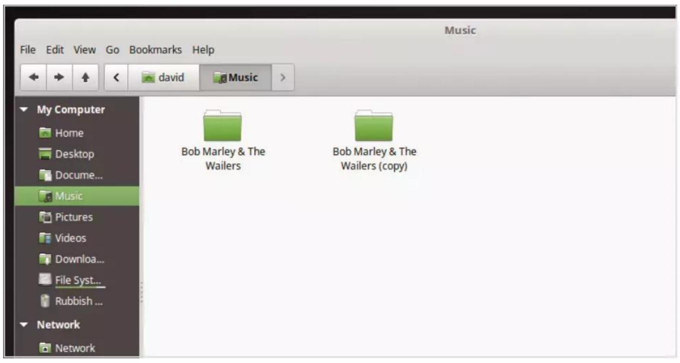

Rubbish Bin
File Edit View Go Bookmarks Help
Rubbish Bin
My Computer
Home
Desktop
Docume...
Music
Pictures
Videos
Downloa...
File Syst...
Rubbish ...
Network
Network
Rubbish Bin
Restore Selected Items
Empty Rubbish Bin
Restore selected items to their original position
16. Punky Reggae Party.mp3
15. Jamming.mp3
14. Exodus.mp3
13. Satisfy My Soul.mp3
12. Redemption Song.mp3
11. Waiting in Vain.mp3
10. I Shot the Sheriff.mp3
09. One Love People Get Ready.mp3
08. Easy Skanking.mp3
07. Stir It Up.mp3
06. Get Up Stand Up.mp3
05. Buffalo Soldier.mp3
04. Three Little Birds.mp3
03. Could You Be Loved.mp3
02. No Woman No Cry [Live].mp3

STEP 2 However, the Terminal doesn't offer that luxury. If you were to access a folder with files within via the Terminal and then enter the command: `rm *.*`, all the files in that folder would be instantly deleted. They won't appear in the Rubbish Bin either, they're gone for good.

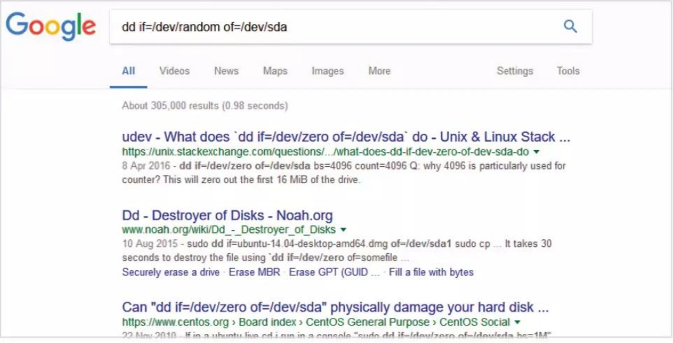

david@david-mint ~/Mu
File Edit View Search Terminal Help

```
david@david-mint ~/Music/Bob Marley & 
01. Is this Love.mp3              09. On
02. No Woman No Cry [Live].mp3    10. I 
03. Could You Be Loved.mp3        11. Wa
04. Three Little Birds.mp3        12. Re
05. Buffalo Soldier.mp3           13. Sa
06. Get Up Stand Up.mp3           14. Ex
07. Stir It Up.mp3                15. Ja
```

STEP 3 Therefore it's always a good idea to work in the Terminal using a two-pronged approach. First, use the desktop to make regular backups of the folders you're working in when in the Terminal. This way, should anything go wrong, there's a quick and handy backup waiting for you.


Music
File Edit View Go Bookmarks Help
david
Music
My Computer
Home
Desktop
Docume...
Music
Pictures
Videos
Downloa...
File Syst...
Rubbish ...
Network
Network
Bob Marley & The Wailers
Bob Marley & The Wailers (copy)

STEP 4 Second, research before blindly entering a command you've seen on the Internet. If you see the command: `sudo dd if=/dev/random of=/dev/sda` and use it in a script, you'll soon come to regret the action as the command will wipe the entire hard drive and fill it with random data. Take a moment to Google the command and see what it does.


Google
All
Videos
News
Maps
Images
More
Settings
Tools

dd if=/dev/random of=/dev/sda

About 305,000 results (0.98 seconds)

udev - What does `dd if=/dev/zero of=/dev/sda` do - Unix & Linux Stack ...
https://unix.stackexchange.com/questions/.../what-does-dd-if-dev-zero-of-dev-sda-do
8 Apr 2016 - dd if=/dev/zero of=/dev/sda bs=4096 count=4096 Q: why 4096 is particularly used for counter? This will zero out the first 16 MiB of the drive.

Dd - Destroyer of Disks - Noah.org
www.noah.org/wiki/Dd_-_Destroyer_of_Disks
10 Aug 2015 - sudo dd if=ubuntu-14.04-desktop-amd64.dmg of=/dev/sda1 sudo cp ... It takes 30 seconds to destroy the file using `dd if=/dev/zero of=somefile ...
Securely erase a drive · Erase MBR · Erase GPT (GUID ... · Fill a file with bytes

Can "dd if=/dev/zero of=/dev/sda" physically damage your hard disk ...
https://www.centos.org › Board index › CentOS General Purpose › CentOS Social

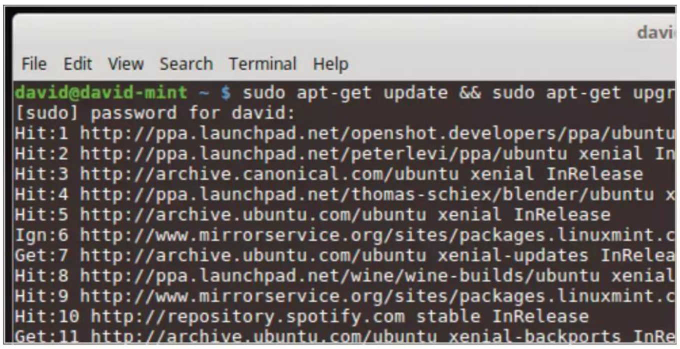
Creating Bash Scripts – Part 1

Eventually, as you advance with Linux Mint, you'll want to start creating your own automated tasks and programs. These are essentially scripts, Bash Shell scripts to be exact, and they work in the same way as a DOS Batch file does, or any other programming language.

GET SCRIPTING

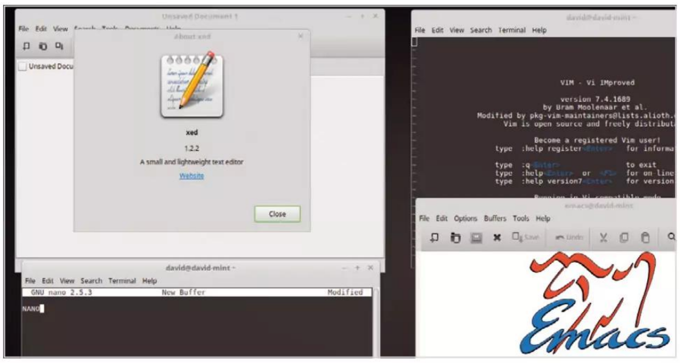
A Bash script is simply a series of commands that Mint will run through to complete a certain task. They can be simple or remarkably complex, it all depends on the situation.

STEP 1 You'll be working within the Terminal and with a text editor throughout the coming pages. There are alternatives to the text editor, which we'll look at in a moment but for the sake of ease, we'll be doing our examples in Xed. Before you begin, however, run through the customary update check: `sudo apt-get update && sudo apt-get upgrade`.


davi
File Edit View Search Terminal Help

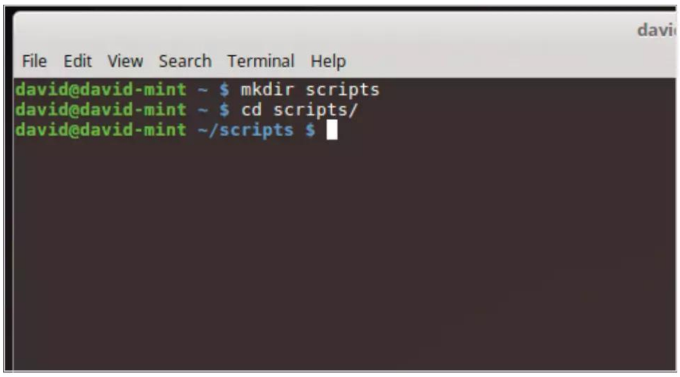
```
david@david-mint ~ $ sudo apt-get update && sudo apt-get upgr
[sudo] password for david:
Hit:1 http://ppa.launchpad.net/openshot.developers/ppa/ubuntu
Hit:2 http://ppa.launchpad.net/peterlevi/ppa/ubuntu xenial In
Hit:3 http://archive.canonical.com/ubuntu xenial InRelease
Hit:4 http://ppa.launchpad.net/thomas-schiex/blender/ubuntu x
Hit:5 http://archive.ubuntu.com/ubuntu xenial InRelease
Ign:6 http://www.mirrorservice.org/sites/packages.linuxmint.c
Get:7 http://archive.ubuntu.com/ubuntu xenial-updates InRelea
Hit:8 http://ppa.launchpad.net/wine/wine-builds/ubuntu xenial
Hit:9 http://www.mirrorservice.org/sites/packages.linuxmint.c
Hit:10 http://repository.spotify.com stable InRelease
Get:11 http://archive.ubuntu.com/ubuntu xenial-backports InRe
```

STEP 2 There are several text editors we can use to create a Bash script: Xed, Vi, Nano, Vim, GNU Emacs and so on. In the end it all comes down to personal preference. Our use of Xed is purely due to making it easier to read the script in the screenshots you see below.


File Edit View
Unsaved Docu
xed
1.2.2
A small and lightweight text editor
Website
Close
david@david-mint ~
File Edit View Search Terminal Help
GNU nano 2.5.3
New Buffer
Modified
NANO
File Edit View Search Terminal Help
File Edit Options Buffers Tools Help
Emacs

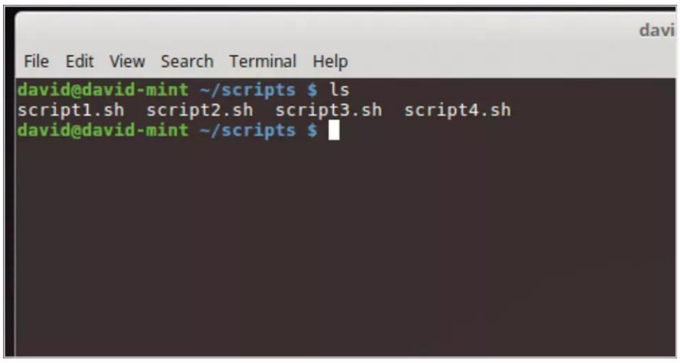
```
VIM - Vi IMproved

version 7.4.1689
by Bram Moolenaar et al.
Modified by pkg-vim-maintainers@lists.alioth.
Vim is open source and freely distribut

Become a registered Vim user!
type  :help register<Enter>   for informa

type  :q<Enter>               to exit
type  :help<Enter>  or  <F1>  for on-line
type  :help version7<Enter>   for version
```

STEP 3 To begin with, and before you start to write any scripts, you need to create a folder where you can put all our scripts into. Start with `mkdir scripts`, and enter the folder `cd scripts/`. This will be our working folder and from here you can create sub-folders if you want of each script you create.


davi
File Edit View Search Terminal Help

```
david@david-mint ~ $ mkdir scripts
david@david-mint ~ $ cd scripts/
david@david-mint ~/scripts $
```

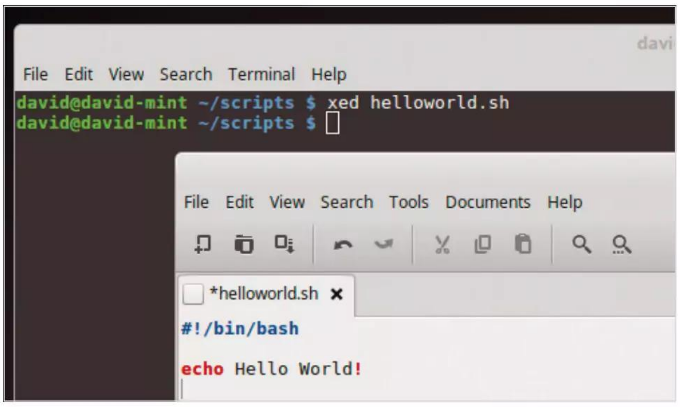
STEP 4 Windows users will be aware that in order for a batch file to work, as in be executed and follow the programming within it, it needs to have a .BAT file extension. Linux is an extension-less operating system but the convention is to give scripts a .sh extension.


davi
File Edit View Search Terminal Help

```
david@david-mint ~/scripts $ ls
script1.sh  script2.sh  script3.sh  script4.sh
david@david-mint ~/scripts $
```

STEP 5

Let's start with a simple script to output something to the Terminal. Enter `xed helloworld.sh`. This will launch Xed and create a file called helloworld.sh. In Xed, enter the following: `#!/bin/bash`, then on a new line: `echo Hello World!`.

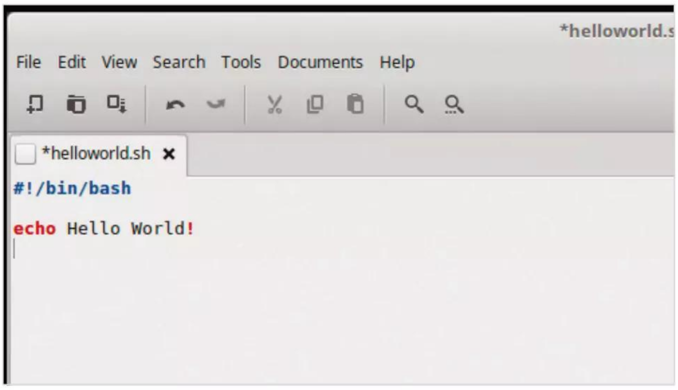

File Edit View Search Terminal Help
File Edit View Search Tools Documents Help
*helloworld.sh

```
david@david-mint ~/scripts $ xed helloworld.sh
david@david-mint ~/scripts $ 
```

```
#!/bin/bash

echo Hello World!
```

STEP 6

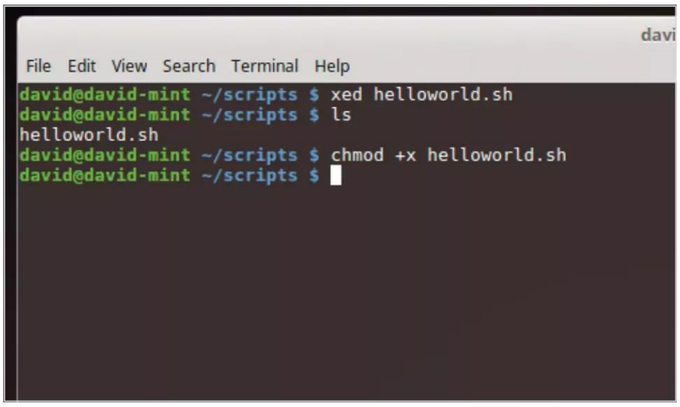
The `#!/bin/bash` line tells the system what Shell you're going to be using, in this case Bash. The hash (#) denotes a comment line, one that is ignored by the system, the exclamation mark (!) means that the comment is bypassed and will force the script to execute the line as a command. This is also known as a Hash-Bang.


*helloworld.s
File Edit View Search Tools Documents Help
*helloworld.sh

```
#!/bin/bash

echo Hello World!
```

STEP 7

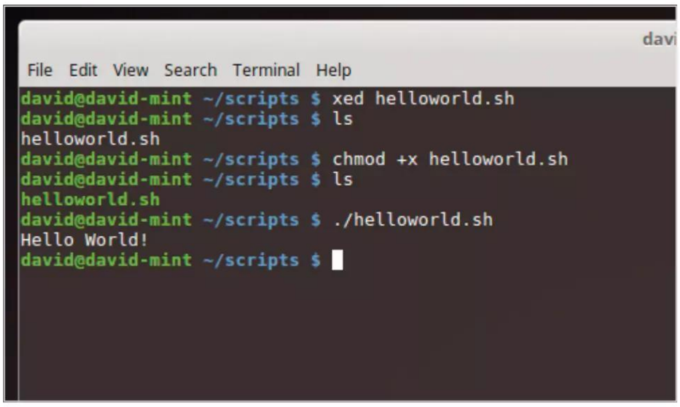
You can save this file, clicking File > Save, and exit back to the Terminal. Entering `ls`, will reveal the script in the folder. To make any script executable, and able to run, you need to modify its permissions. Do this with `chmod +x helloworld.sh`. You need to do this with every script you create.


File Edit View Search Terminal Help

```
david@david-mint ~/scripts $ xed helloworld.sh
david@david-mint ~/scripts $ ls
helloworld.sh
david@david-mint ~/scripts $ chmod +x helloworld.sh
david@david-mint ~/scripts $ 
```

STEP 8

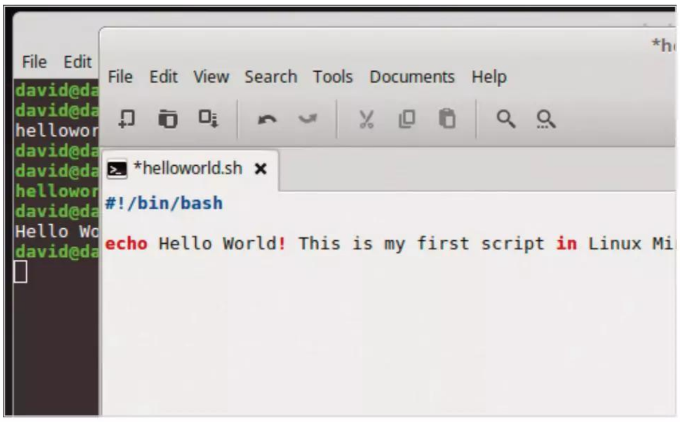
When you enter `ls` again, you can see that the helloworld.sh script has now turned from being white to green, meaning that it's now an executable file. To run the script, in other words make it do the things you've typed into it, enter: `./helloworld.sh`.


File Edit View Search Terminal Help

```
david@david-mint ~/scripts $ xed helloworld.sh
david@david-mint ~/scripts $ ls
helloworld.sh
david@david-mint ~/scripts $ chmod +x helloworld.sh
david@david-mint ~/scripts $ ls
helloworld.sh
david@david-mint ~/scripts $ ./helloworld.sh
Hello World!
david@david-mint ~/scripts $ 
```

STEP 9

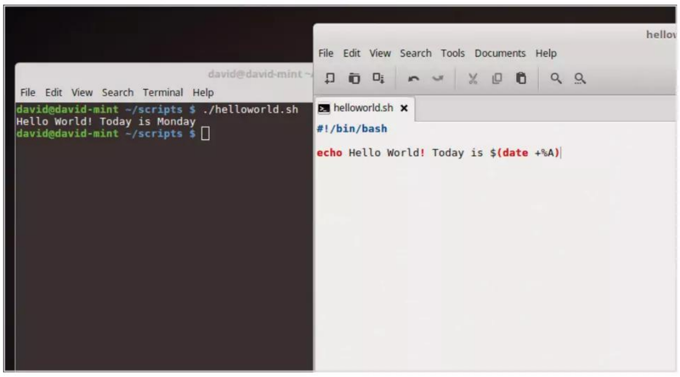
Although it's not terribly exciting, the words 'Hello World!' should now be displayed in the Terminal. The echo command is responsible for outputting the words after it in the Terminal, as we move on you can make the echo command output to other sources.


File Edit
File Edit View Search Tools Documents Help
*helloworld.sh

```
#!/bin/bash

echo Hello World! This is my first script in Linux Mi
```

STEP 10

Think of echo as the old BASIC Print command. It displays either text, numbers or any variables that are stored in the system, such as the current system date. Try this example: `echo Hello World! Today is $(date +%A)`. The $(date +%A) is calling the system variable that stores the current day of the week.


File Edit View Search Terminal Help
File Edit View Search Tools Documents Help
helloworld.sh

```
david@david-mint ~/scripts $ ./helloworld.sh
Hello World! Today is Monday
david@david-mint ~/scripts $ 
```

```
#!/bin/bash

echo Hello World! Today is $(date +%A)
```

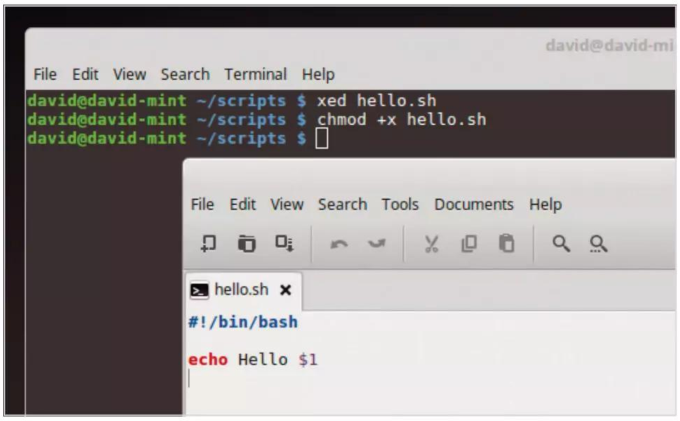
Creating Bash Scripts – Part 2

Previously we looked at creating your first Bash script, Hello World, and adding a system variable. Now you can expand these and see what you can do when you start to play around with creating your own unique variables.

VARIABLES

Just as in every other programming language a Bash script can store and call certain variables from the system, either generic or user created.

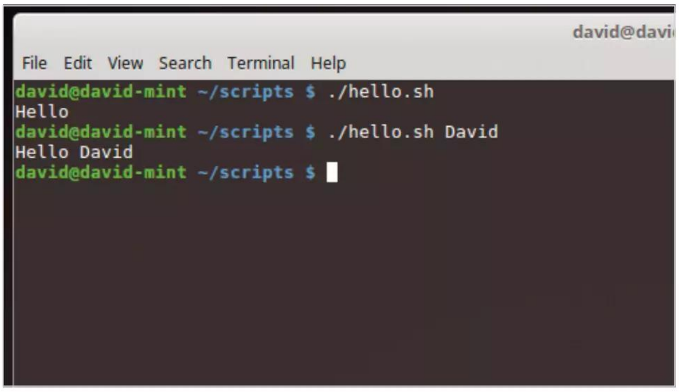
STEP 1 Let's start by creating a new script called hello.sh; `xed hello.sh`. In it enter: `#!/bin/bash`, then, echo `Hello $1`. Save the file and exit Xed. Back in the Terminal make the script executable with: `chmod +x hello.sh`.


david@david-mi
File Edit View Search Terminal Help
File Edit View Search Tools Documents Help
hello.sh

```
david@david-mint ~/scripts $ xed hello.sh
david@david-mint ~/scripts $ chmod +x hello.sh
david@david-mint ~/scripts $ 
```

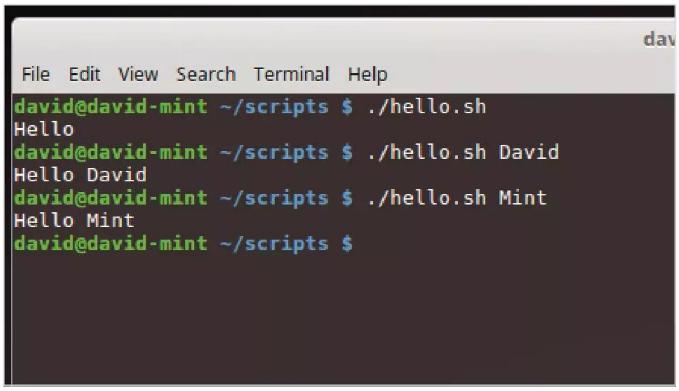
```
#!/bin/bash

echo Hello $1
```

STEP 2 As the script is now executable, run it with `./hello.sh`. Now, as you probably expected a simple 'Hello' is displayed in the Terminal. However, if you then issue the command with a variable, it begins to get interesting. For example, try `./hello.sh David`.


david@davi
File Edit View Search Terminal Help

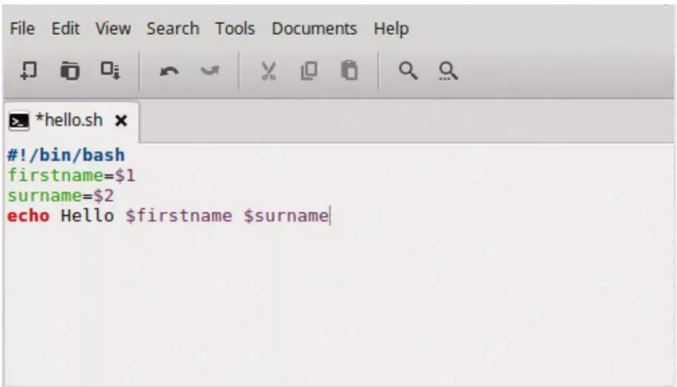
```
david@david-mint ~/scripts $ ./hello.sh
Hello
david@david-mint ~/scripts $ ./hello.sh David
Hello David
david@david-mint ~/scripts $ 
```

STEP 3 The output now will be Hello David. This is because Bash automatically assigns variables for the user, which are then held and passed to the script. So the variable '$1' now holds 'David'. You can change the variable by entering something different: `./hello.sh Mint`.


dav
File Edit View Search Terminal Help

```
david@david-mint ~/scripts $ ./hello.sh
Hello
david@david-mint ~/scripts $ ./hello.sh David
Hello David
david@david-mint ~/scripts $ ./hello.sh Mint
Hello Mint
david@david-mint ~/scripts $
```

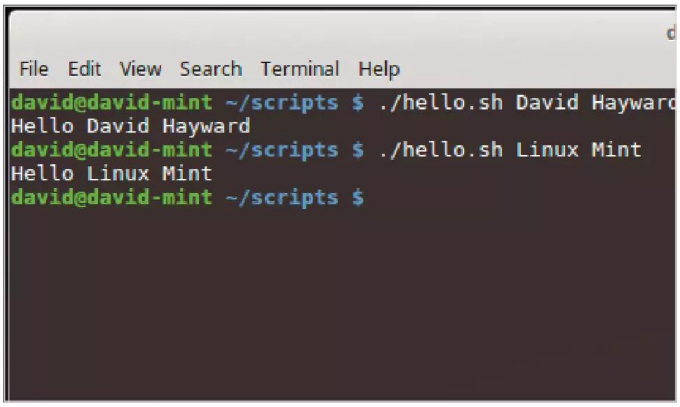
STEP 4 You can even rename variables. Modify the hello.sh script with the following: `firstname=$1, surname=$2, echo Hello $firstname $surname`. Putting each statement on a new line. Save the script and exit back into the Terminal.


File Edit View Search Tools Documents Help
*hello.sh

```
#!/bin/bash
firstname=$1
surname=$2
echo Hello $firstname $surname
```

STEP 5

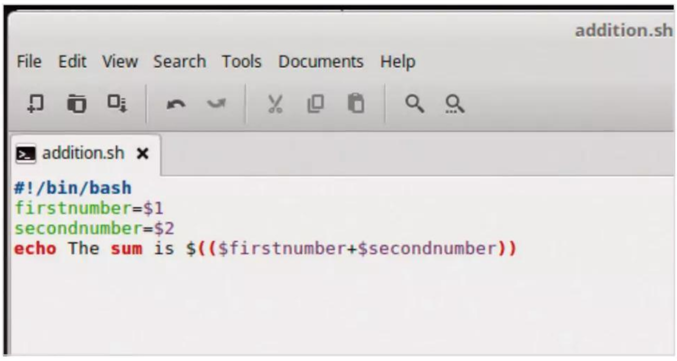
When you run the script now you can use two custom variables: `./hello.sh David Hayward`. Naturally change the two variables with your own name; unless you're also called David Hayward. At the moment we're just printing the contents, so let's expand the two-variable use a little.


d
File Edit View Search Terminal Help

```
david@david-mint ~/scripts $ ./hello.sh David Haywar
Hello David Hayward
david@david-mint ~/scripts $ ./hello.sh Linux Mint
Hello Linux Mint
david@david-mint ~/scripts $
```

STEP 6

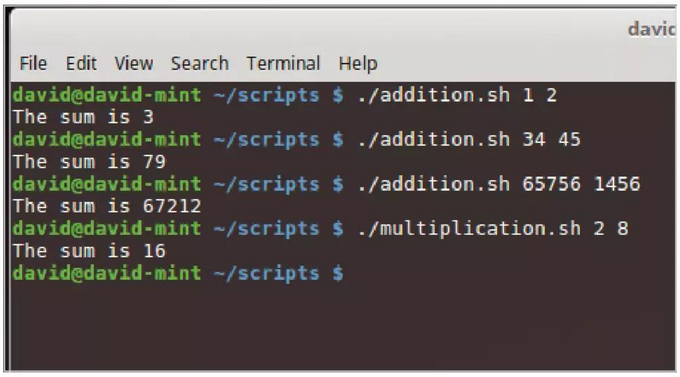
Create a new script called `addition.sh`, using the same format as the hello.sh script, but changing the variable names. Here we've added `firstnumber` and `secondnumber`, and used the echo command to output some simple arithmetic by placing an integer expression, `echo The sum is $(($firstnumber+$secondnumber))`. Save the script, and make it executable `(chmod +x addition.sh)`.


addition.sh
File Edit View Search Tools Documents Help
addition.sh ×

```
#!/bin/bash
firstnumber=$1
secondnumber=$2
echo The sum is $(($firstnumber+$secondnumber))
```

STEP 7

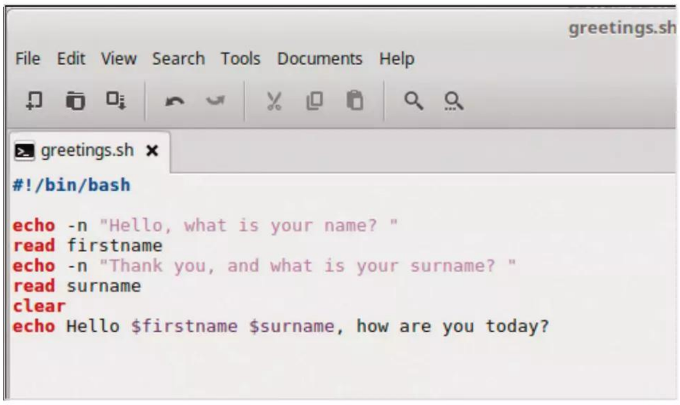
When you now run the addition.sh script we can enter two numbers: `./addition.sh 1 2`. The result will hopefully be 3, with the Terminal displaying 'The sum is 3'. Try it with a few different numbers and see what happens. See also if you can alter the script and rename it do multiplication, and subtraction.


davic
File Edit View Search Terminal Help

```
david@david-mint ~/scripts $ ./addition.sh 1 2
The sum is 3
david@david-mint ~/scripts $ ./addition.sh 34 45
The sum is 79
david@david-mint ~/scripts $ ./addition.sh 65756 1456
The sum is 67212
david@david-mint ~/scripts $ ./multiplication.sh 2 8
The sum is 16
david@david-mint ~/scripts $
```

STEP 8

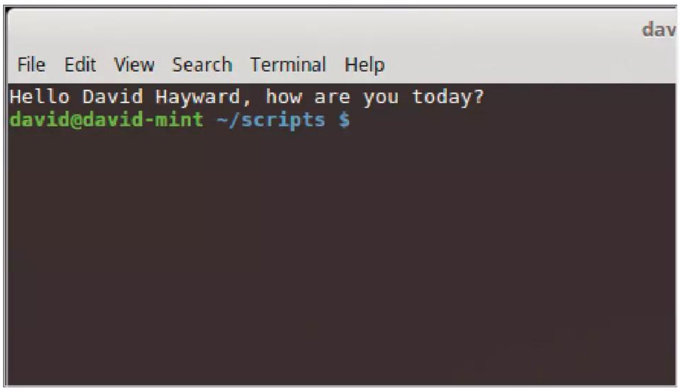
Let's expand things further. Create a new script called `greetings.sh`. Enter the scripting as below in the screenshot, save it and make it executable with the `chmod` command. You can see that there are a few new additions to the script now.


greetings.sh
File Edit View Search Tools Documents Help
greetings.sh ×

```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
echo Hello $firstname $surname, how are you today?
```

STEP 9

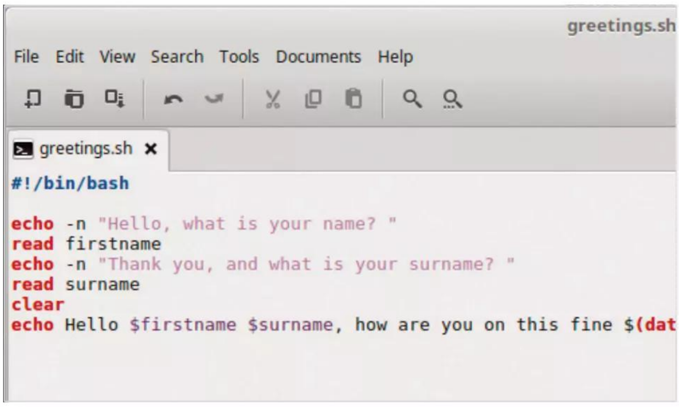
We've added a `-n` to the echo command here which will leave the cursor on the same line as the question, instead of a new line. The `read` command stores the users' input as the variables firstname and surname, to then read back later in the last `echo` line. And the `clear` command clears the screen.


dav
File Edit View Search Terminal Help

```
Hello David Hayward, how are you today?
david@david-mint ~/scripts $
```

STEP 10

As a final addition, let's include the date variable we used in the last section. Amend the last line of the script to read: `echo Hello $firstname $surname, how are you on this fine $(date +%A)?`. The output should display the current day of the week, calling it from a system variable.


greetings.sh
File Edit View Search Tools Documents Help
greetings.sh ×

```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
echo Hello $firstname $surname, how are you on this fine $(dat
```

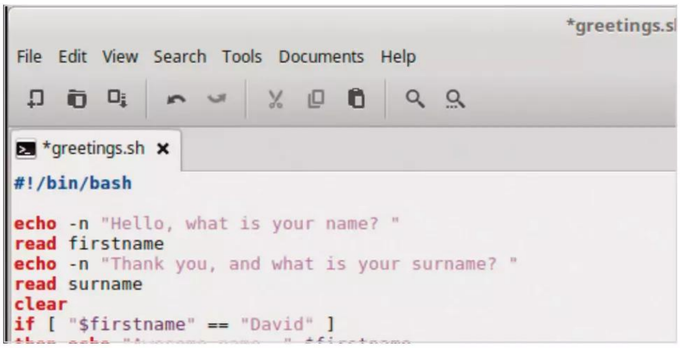
Creating Bash Scripts – Part 3

In the previous pages we looked at some very basic Bash scripting, which involved outputting text to the screen, getting a user's input, storing it and outputting that to the screen; as well as including a system variable using the Date command. Now let's combine what you've achieved so far and introduce Loops.

IF, THEN, ELSE

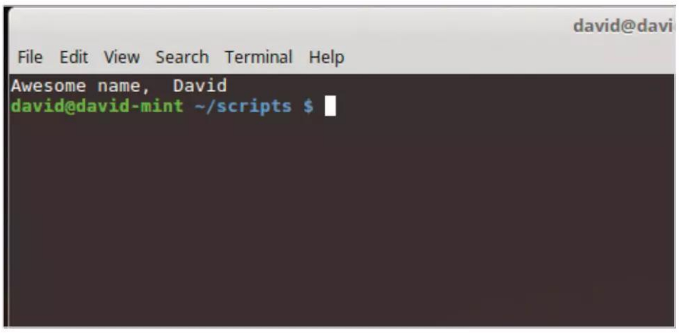
With most programming structures there will come a time where you need to loop through the commands you've entered to create better functionality, and ultimately a better program.

STEP 1 Let's look at the If, Then and Else statements now, which when executed correctly, compare a set of instructions and simply work out that IF something is present, THEN do something, ELSE do something different. Create a new script called `greeting2.sh` and enter the text in the screenshot below into it.



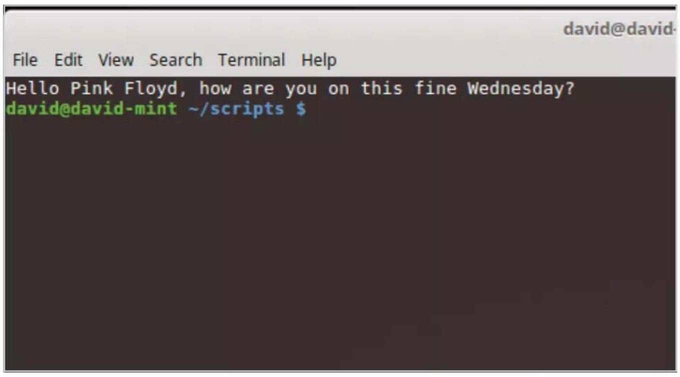
```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
if [ "$firstname" == "David" ]
```

STEP 2 Greeting2.sh is a copy of greeting.sh but with a slight difference. Here we've added a loop starting at the if statement. This means, IF the variable entered is equal to David the next line, THEN, is the reaction to what happens, in this case it will output to the screen 'Awesome name,' followed by the variable (which is David).



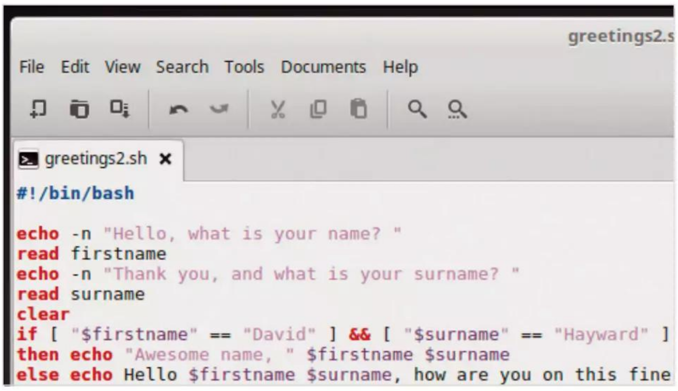
```
Awesome name,  David
david@david-mint ~/scripts $
```

STEP 3 The next line, ELSE, is what happens if the variable doesn't equal 'David'. In this case it simply outputs to the screen the now familiar 'Hello…'. The last line, the FI statement, is the command that will end the loop. If you have an If command without a Fi command, then you get an error.



```
Hello Pink Floyd, how are you on this fine Wednesday?
david@david-mint ~/scripts $
```

STEP 4 You can obviously play around with the script a little, changing the name variable that triggers a response; or maybe even issuing a response where the first name and surname variables match a specific variable.

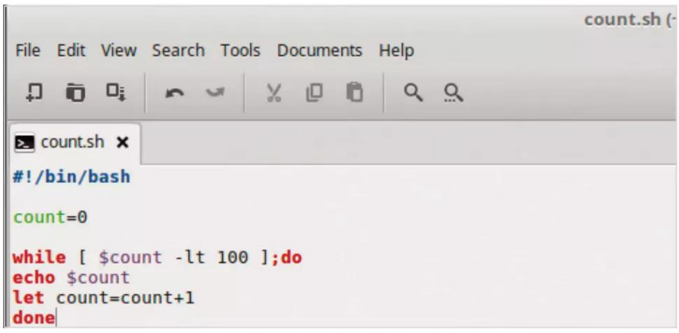


```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
if [ "$firstname" == "David" ] && [ "$surname" == "Hayward" ]
then echo "Awesome name, " $firstname $surname
else echo Hello $firstname $surname, how are you on this fine
```

MORE LOOPING

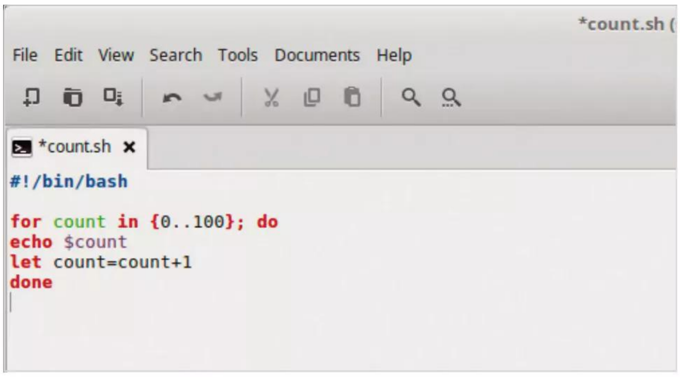
You can loop over data using the FOR, WHILE and UNTIL statements. These can be handy if you're batch naming, copying or running a script where a counter is needed.

STEP 1 Create a new script called `count.sh`. Enter the text in the screenshot below, save it and make it executable. This creates the variable 'count' which at the beginning of the script equals zero. Then start the WHILE loop, which WHILE count is less than (the LT part) 100 will print the current value of count in the echo command.


count.sh (
File Edit View Search Tools Documents Help
count.sh

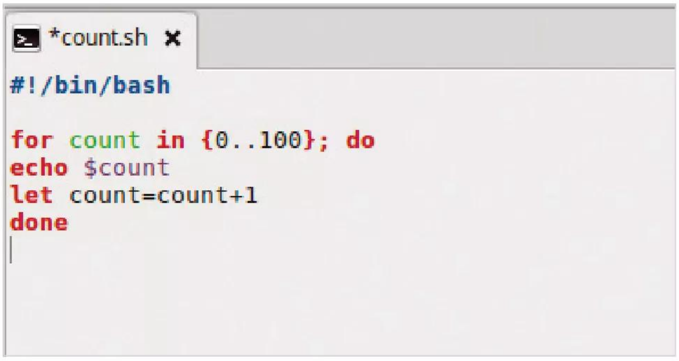
```
#!/bin/bash

count=0

while [ $count -lt 100 ];do
echo $count
let count=count+1
done
```

STEP 2 Executing the count.sh script will result in the numbers 0 to 99 listing down the Terminal screen; when it reaches 100 the script will end. Modifying the script with the FOR statement, makes it work in much the same way. To use it in our script, enter the text from the screenshot into the count.sh script.


*count.sh (
File Edit View Search Tools Documents Help
*count.sh

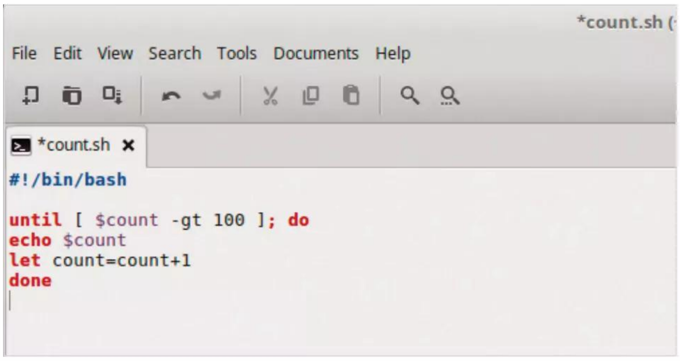
```
#!/bin/bash

for count in {0..100}; do
echo $count
let count=count+1
done
```

STEP 3 The addition we have here is: `for count in {0..100}; do`. Which means: FOR the variable 'count' IN the numbers from zero to one hundred, then start the loop. The rest of the script is the same. Run this script, and the same output should appear in the Terminal.



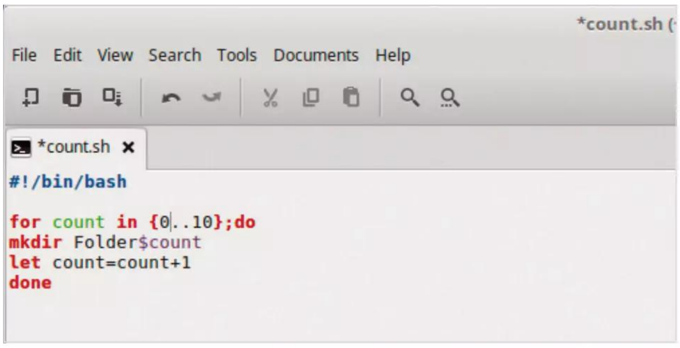
```
#!/bin/bash

for count in {0..100}; do
echo $count
let count=count+1
done
```

STEP 4 The UNTIL loop works much the same way as the WHILE loop only, more often than not, in reverse. So our counting to a hundred, using `UNTIL, would be: until [ $count -gt 100 ]; do`. The difference being, UNTIL count is greater than (the gt part) one hundred, keep on looping.


*count.sh (
File Edit View Search Tools Documents Help
*count.sh

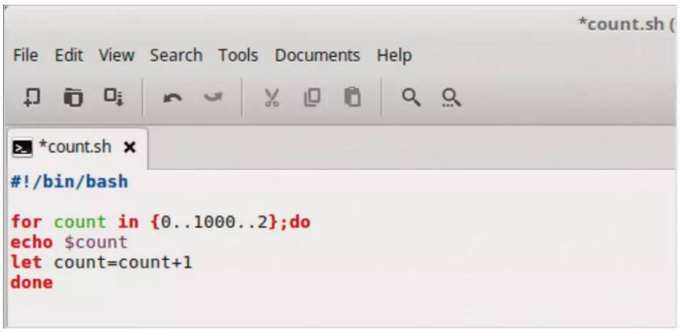
```
#!/bin/bash

until [ $count -gt 100 ]; do
echo $count
let count=count+1
done
```

STEP 5 You're not limited to numbers zero to one hundred. You can, within the loop, have whatever set of commands you like and execute them as many times as you want the loop to run for. Renaming a million files, creating fifty folders etc. For example, this script will create ten folders named folder1 through to folder10 using the FOR loop.


*count.sh (
File Edit View Search Tools Documents Help
*count.sh

```
#!/bin/bash

for count in {0..10};do
mkdir Folder$count
let count=count+1
done
```

STEP 6 Using the FOR statement once more, we can execute the counting sequence by manipulating the {0..100} part. This section of the code actually means {START..END..INCREMENT}, if there's no increment then it's just a single digit up to the END. For example, we could get the loops to count up to 1000 in two's with: `for count in {0..1000..2}; do`.


*count.sh (
File Edit View Search Tools Documents Help
*count.sh

```
#!/bin/bash

for count in {0..1000..2};do
echo $count
let count=count+1
done
```

Creating Bash Scripts – Part 4

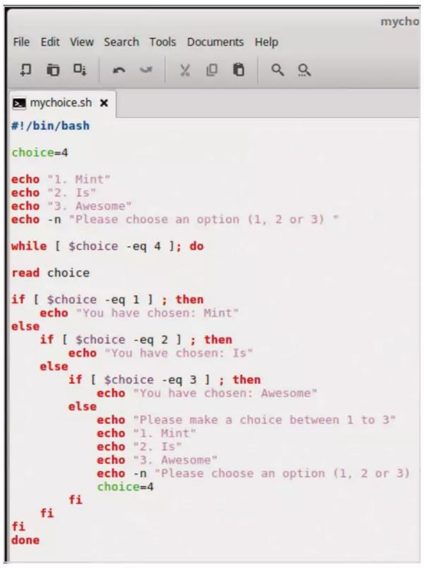
You've encountered user interaction with your scripts, asking what the user's name is and so on. You've also looked at creating loops within the script to either count or simply do something several times. Let's combine and expand some more.

CHOICES AND LOOPS

Let's bring in another command, CHOICE, along with some nested IF and ELSE statements. Start by creating a new script called `mychoice.sh`.

STEP 1

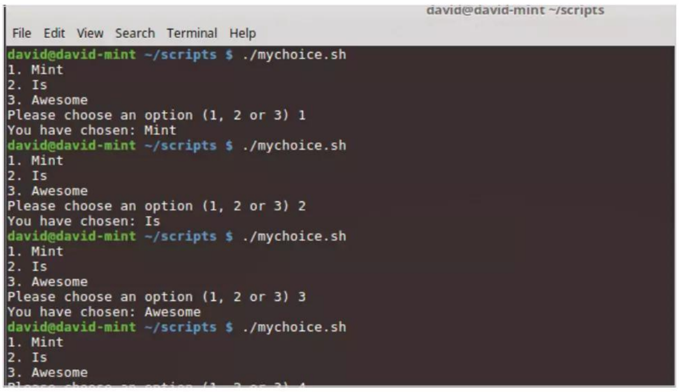
The mychoice.sh script is beginning to look a lot more complex. What we have here is a list of four choices, with three possible options. The options: Mint, Is, and Awesome will be displayed if the user presses the correct option key. If not, then the menu will reappear, the fourth choice.



```
#!/bin/bash

choice=4

echo "1. Mint"
echo "2. Is"
echo "3. Awesome"
echo -n "Please choose an option (1, 2 or 3) "

while [ $choice -eq 4 ]; do

read choice

if [ $choice -eq 1 ] ; then
    echo "You have chosen: Mint"
else
    if [ $choice -eq 2 ] ; then
        echo "You have chosen: Is"
    else
        if [ $choice -eq 3 ] ; then
            echo "You have chosen: Awesome"
        else
            echo "Please make a choice between 1 to 3"
            echo "1. Mint"
            echo "2. Is"
            echo "3. Awesome"
            echo -n "Please choose an option (1, 2 or 3)
            choice=4
        fi
    fi
fi
done
```

STEP 2

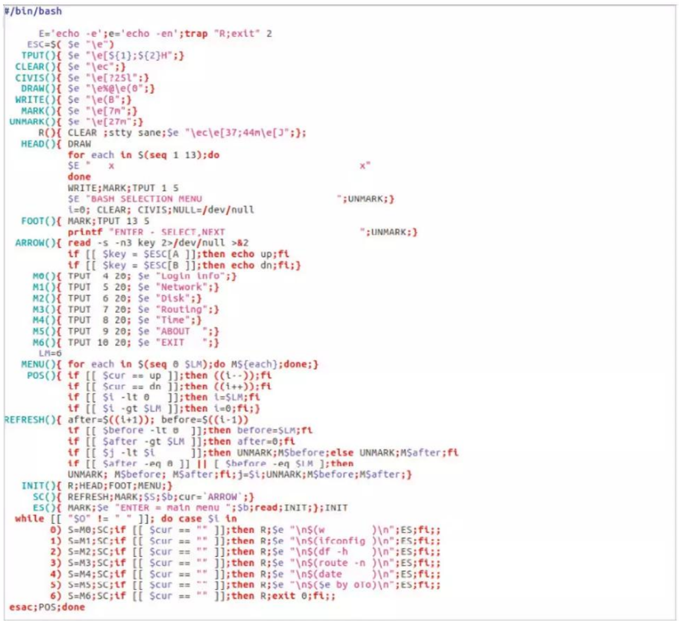
If you follow the script through you soon get the hang of what's going on, based on what we've already covered. WHILE, IF, and ELSE, with the FI closing loop statement will run through the options and bring you back to the start if you pick the wrong option.


david@david-mint ~/scripts
File Edit View Search Terminal Help

```
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
Please choose an option (1, 2 or 3) 1
You have chosen: Mint
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
Please choose an option (1, 2 or 3) 2
You have chosen: Is
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
Please choose an option (1, 2 or 3) 3
You have chosen: Awesome
david@david-mint ~/scripts $ ./mychoice.sh
1. Mint
2. Is
3. Awesome
```

STEP 3

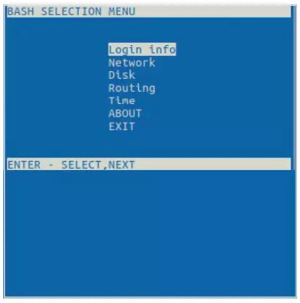
You can, of course, increase the number of choices but you need to make sure that you match the number of choices to the number of IF statements. The script can quickly become a very busy screen to look at. This lengthy script is another way of displaying a menu, this time with a fancy colour scheme too.



```
#/bin/bash

      E='echo -e';e='echo -en';trap "R;exit" 2
    ESC=$( $e "\e")
   TPUT(){ $e "\e[${1};${2}H";}
  CLEAR(){ $e "\ec";}
  CIVIS(){ $e "\e[?25l";}
   DRAW(){ $e "\e%@\e(0";}
  WRITE(){ $e "\e(B";}
   MARK(){ $e "\e[7m";}
 UNMARK(){ $e "\e[27m";}
      R(){ CLEAR ;stty sane;$e "\ec\e[37;44m\e[J";};
   HEAD(){ DRAW
           for each in $(seq 1 13);do
           $E "   x                                          x"
           done
           WRITE;MARK;TPUT 1 5
           $E "BASH SELECTION MENU                       ";UNMARK;}
           i=0; CLEAR; CIVIS;NULL=/dev/null
   FOOT(){ MARK;TPUT 13 5
           printf "ENTER - SELECT,NEXT                       ";UNMARK;}
  ARROW(){ read -s -n3 key 2>/dev/null >&2
           if [[ $key = $ESC[A ]];then echo up;fi
           if [[ $key = $ESC[B ]];then echo dn;fi;}
     M0(){ TPUT  4 20; $e "Login info";}
     M1(){ TPUT  5 20; $e "Network";}
     M2(){ TPUT  6 20; $e "Disk";}
     M3(){ TPUT  7 20; $e "Routing";}
     M4(){ TPUT  8 20; $e "Time";}
     M5(){ TPUT  9 20; $e "ABOUT  ";}
     M6(){ TPUT 10 20; $e "EXIT   ";}
      LM=6
   MENU(){ for each in $(seq 0 $LM);do M${each};done;}
    POS(){ if [[ $cur == up ]];then ((i--));fi
           if [[ $cur == dn ]];then ((i++));fi
           if [[ $i -lt 0   ]];then i=$LM;fi
           if [[ $i -gt $LM ]];then i=0;fi;}
REFRESH(){ after=$((i+1)); before=$((i-1))
           if [[ $before -lt 0  ]];then before=$LM;fi
           if [[ $after -gt $LM ]];then after=0;fi
           if [[ $j -lt $i      ]];then UNMARK;M$before;else UNMARK;M$after;fi
           if [[ $after -eq 0 ]] || [ $before -eq $LM ];then
           UNMARK; M$before; M$after;fi;j=$i;UNMARK;M$before;M$after;}
   INIT(){ R;HEAD;FOOT;MENU;}
     SC(){ REFRESH;MARK;$S;$b;cur=`ARROW`;}
     ES(){ MARK;$e "ENTER = main menu ";$b;read;INIT;};INIT
  while [[ "$O" != " " ]]; do case $i in
        0) S=M0;SC;if [[ $cur == "" ]];then R;$e "\n$(w        )\n";ES;fi;;
        1) S=M1;SC;if [[ $cur == "" ]];then R;$e "\n$(ifconfig )\n";ES;fi;;
        2) S=M2;SC;if [[ $cur == "" ]];then R;$e "\n$(df -h    )\n";ES;fi;;
        3) S=M3;SC;if [[ $cur == "" ]];then R;$e "\n$(route -n )\n";ES;fi;;
        4) S=M4;SC;if [[ $cur == "" ]];then R;$e "\n$(date     )\n";ES;fi;;
        5) S=M5;SC;if [[ $cur == "" ]];then R;$e "\n$($e by oTo)\n";ES;fi;;
        6) S=M6;SC;if [[ $cur == "" ]];then R;exit 0;fi;;
 esac;POS;done
```

STEP 4

You can use the arrow keys and Enter in the menu setup in the script. Each choice is an external command that feeds back various information. Play around with the commands and choices, and see what you can come up with. It's a bit beyond what we've looked at but it gives a good idea of what can be achieved.

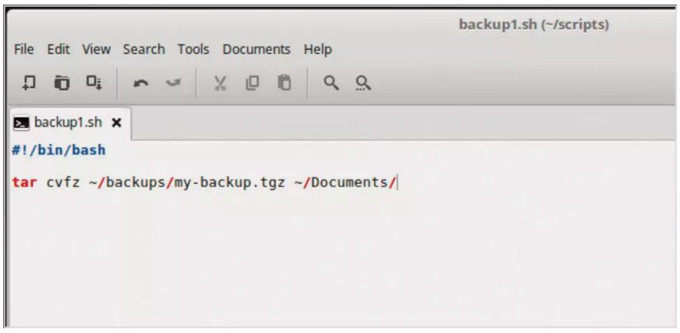


```
BASH SELECTION MENU

                   Login info
                   Network
                   Disk
                   Routing
                   Time
                   ABOUT
                   EXIT

ENTER - SELECT,NEXT
```

CREATING A BACKUP TASK SCRIPT

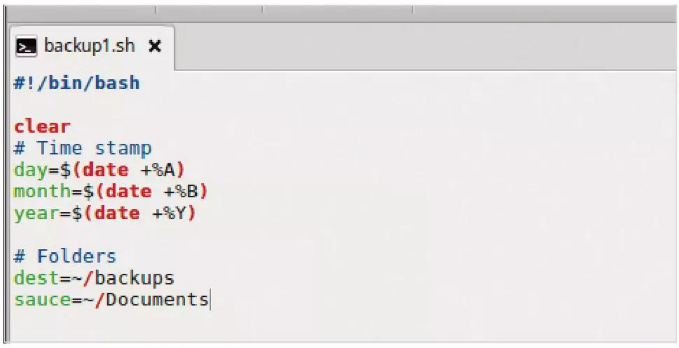
One of the most well used examples of Bash scripting is the creation of a backup routine, one that automates the task as well as adding some customisations along the way.

STEP 1 A very basic backup script would look something along the lines of: `#!/bin/bash, then, tar cvfz ~/backups/my-backup.tgz ~/Documents/`. This will create a compressed file backup of the ~/Documents folder, with everything in it, and put it in a folder called /backups with the name my-backup.tgz.



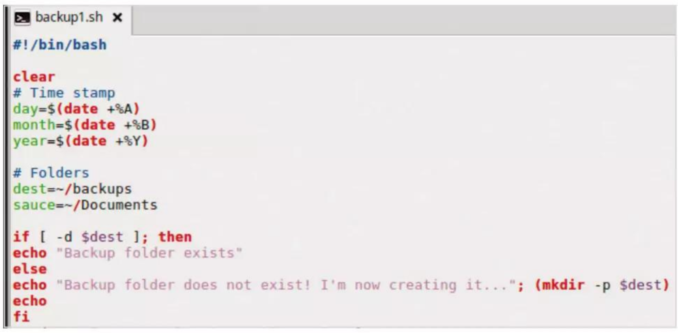
```
#!/bin/bash

tar cvfz ~/backups/my-backup.tgz ~/Documents/
```

STEP 2 While perfectly fine, we can make the simple script a lot more interactive. Let's begin with defining some variables. Enter the text in the screenshot into a new backup.sh script. Notice that we've misspelt 'source' as 'sauce', this is because there's already a built-in command called 'source' hence the different spelling on our part.



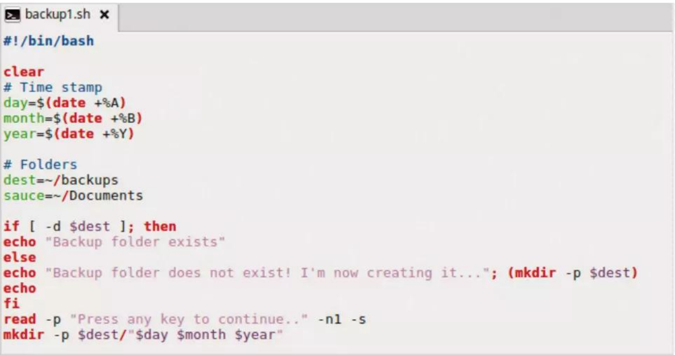
```
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents
```

STEP 3 The previous script entries allowed you to create a Time Stamp, so you know when the backup was taken. You also created a 'dest' variable, which is the folder where the backup file will be created (~/backups). You can now add a section of code to first check if the ~/backups folder exits, if not, then it creates one.



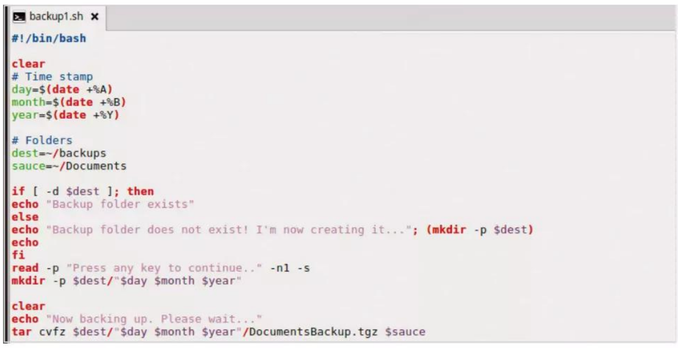
```
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
```

STEP 4 Once the ~/backups folder is created, we can now create a new subfolder within it based on the Time Stamp variables you set up at the beginning. Add `mkdir -p $dest/"$day $month $year"`. It's in here that you put the backup file relevant to that day/month/year.



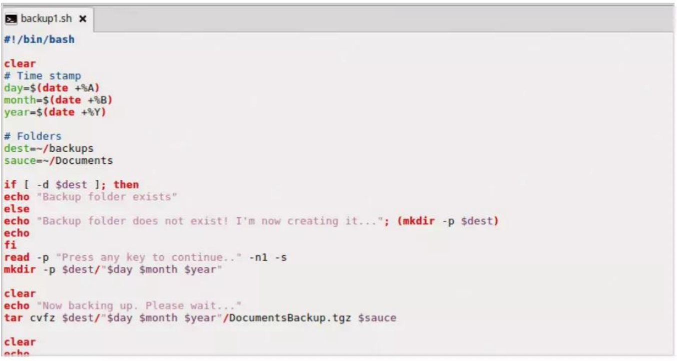
```
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
read -p "Press any key to continue.." -nl -s
mkdir -p $dest/"$day $month $year"
```

STEP 5 With everything in place, you can now enter the actual backup routine, based on the Tar command from Step 5. Combined with the variables, you have: `tar cvfz $dest/"$day $month $year"/DocumentsBackup.tgz $sauce`. In the screenshot, we added a handy "Now backing up..." echo command.



```
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
read -p "Press any key to continue.." -nl -s
mkdir -p $dest/"$day $month $year"

clear
echo "Now backing up. Please wait..."
tar cvfz $dest/"$day $month $year"/DocumentsBackup.tgz $sauce
```

STEP 6 Finally, you can add a friendly message: `echo "Backup complete. All done..."`. The completed script isn't too over-complex and it can be easily customised to include any folder within your Home area, as well as the entire Home area itself.



```
#!/bin/bash

clear
# Time stamp
day=$(date +%A)
month=$(date +%B)
year=$(date +%Y)

# Folders
dest=~/backups
sauce=~/Documents

if [ -d $dest ]; then
echo "Backup folder exists"
else
echo "Backup folder does not exist! I'm now creating it..."; (mkdir -p $dest)
echo
fi
read -p "Press any key to continue.." -nl -s
mkdir -p $dest/"$day $month $year"

clear
echo "Now backing up. Please wait..."
tar cvfz $dest/"$day $month $year"/DocumentsBackup.tgz $sauce

clear
echo
```

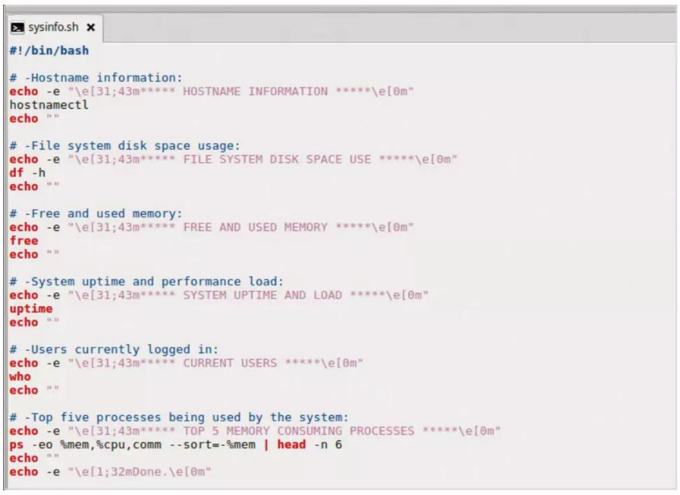
Creating Bash Scripts – Part 5

The backup script we looked at previously can be further amended to incorporate choices; or in other words, user-interaction with regards to where the backup file will be copied to and so on. Automating tasks is one of the main benefits of Bash scripting, a simple script can help you out in many ways.

EASY AUTOMATION AND HANDY SCRIPTS

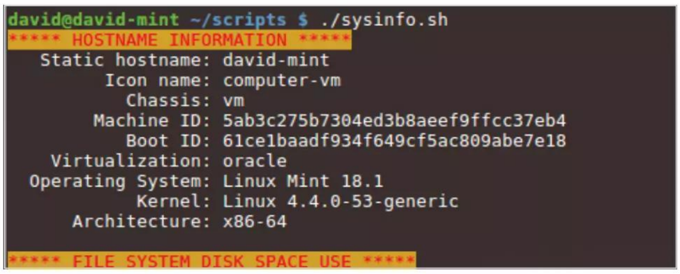
Are you entering line after line of commands to retrieve system information, find a file or rename a batch of files? A script is a better answer.

STEP 1 Let's start by creating a script to help display the Mint system information; always a handy thing to have. Create a new script called `sysinfo.sh` and enter the following into Xed, or the text editor of your choice.



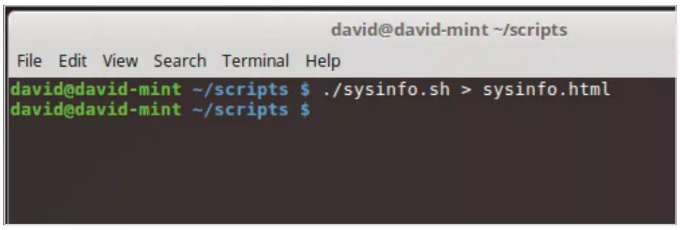
```
#!/bin/bash

# -Hostname information:
echo -e "\e[31;43m***** HOSTNAME INFORMATION *****\e[0m"
hostnamectl
echo ""

# -File system disk space usage:
echo -e "\e[31;43m***** FILE SYSTEM DISK SPACE USE *****\e[0m"
df -h
echo ""

# -Free and used memory:
echo -e "\e[31;43m***** FREE AND USED MEMORY *****\e[0m"
free
echo ""

# -System uptime and performance load:
echo -e "\e[31;43m***** SYSTEM UPTIME AND LOAD *****\e[0m"
uptime
echo ""

# -Users currently logged in:
echo -e "\e[31;43m***** CURRENT USERS *****\e[0m"
who
echo ""

# -Top five processes being used by the system:
echo -e "\e[31;43m***** TOP 5 MEMORY CONSUMING PROCESSES *****\e[0m"
ps -eo %mem,%cpu,comm --sort=-%mem | head -n 6
echo ""
echo -e "\e[1;32mDone.\e[0m"
```

STEP 2 We've included a couple of extra commands in this script. The first is the -e extension for echo, this means it'll enable echo interpretation of additional instances of a new line, as well as other special characters. The proceeding '31;43m' element enables colour for foreground and background.



```
david@david-mint ~/scripts $ ./sysinfo.sh
***** HOSTNAME INFORMATION *****
   Static hostname: david-mint
         Icon name: computer-vm
           Chassis: vm
        Machine ID: 5ab3c275b7304ed3b8aeef9ffcc37eb4
           Boot ID: 61ce1baadf934f649cf5ac809abe7e18
    Virtualization: oracle
  Operating System: Linux Mint 18.1
            Kernel: Linux 4.4.0-53-generic
      Architecture: x86-64

***** FILE SYSTEM DISK SPACE USE *****
```

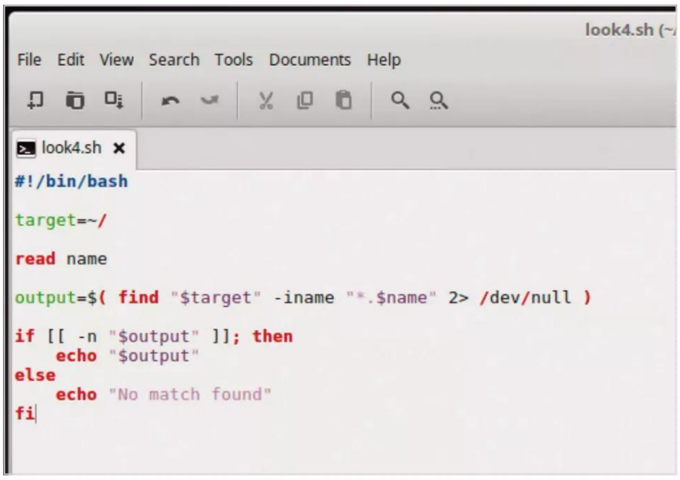
STEP 3 Each of the sections runs a different Terminal command, outputting the results under the appropriate heading. You can include a lot more, such as the current aliases being used in the system, the current time and date and so on. Plus, you could also pipe all that information into a handy HTML file, ready to be viewed in a browser.



david@david-mint ~/scripts
File Edit View Search Terminal Help

```
david@david-mint ~/scripts $ ./sysinfo.sh > sysinfo.html
david@david-mint ~/scripts $
```

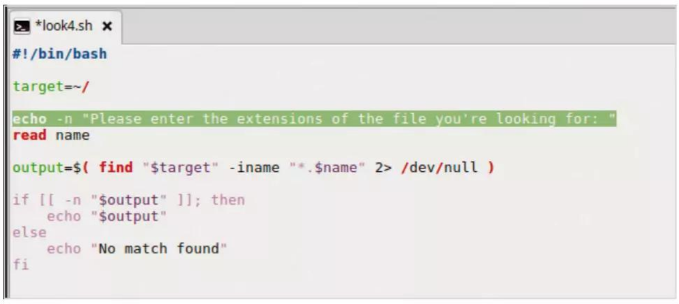
STEP 4 Although there are simple Terminal commands to help you look for a particular file or folder, it's often more fun to create a script to help you. Plus, you can use that script for other non-technical users. Create a new script called `look4.sh`, entering the content from the screenshot below.



```
#!/bin/bash

target=~/

read name

output=$( find "$target" -iname "*.$name" 2> /dev/null )

if [[ -n "$output" ]]; then
    echo "$output"
else
    echo "No match found"
fi
```

STEP 5

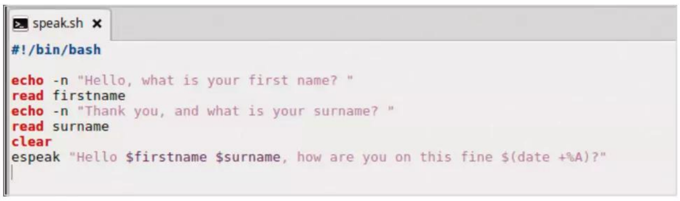
When executed the script waits for input from the user, in this case the file extension, such as jpg, mp4 and so on. It's not very friendly though. Let's make it a little friendlier. Add an echo, with: `echo -n "Please enter the extension of the file you're looking for: "`, just before the read command.



```
*look4.sh ✖
#!/bin/bash

target=~/

echo -n "Please enter the extensions of the file you're looking for: "
read name

output=$( find "$target" -iname "*.$name" 2> /dev/null )

if [[ -n "$output" ]]; then
    echo "$output"
else
    echo "No match found"
fi
```

STEP 6

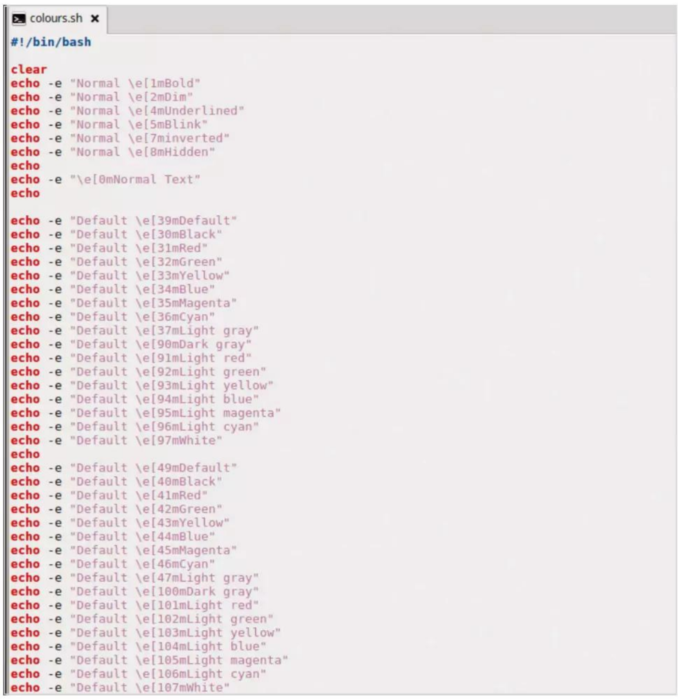
Here's an interesting, fun kind of script using the app espeak. Install espeak with `sudo apt-get install espeak`, then enter the text below into a new script called `speak.sh`. As you can see it's a rehash of the first greeting script we ran. Only this time, it uses the variables in the espeak output.



```
speak.sh ✖
#!/bin/bash

echo -n "Hello, what is your first name? "
read firstname
echo -n "Thank you, and what is your surname? "
read surname
clear
espeak "Hello $firstname $surname, how are you on this fine $(date +%A)?"
```

STEP 7

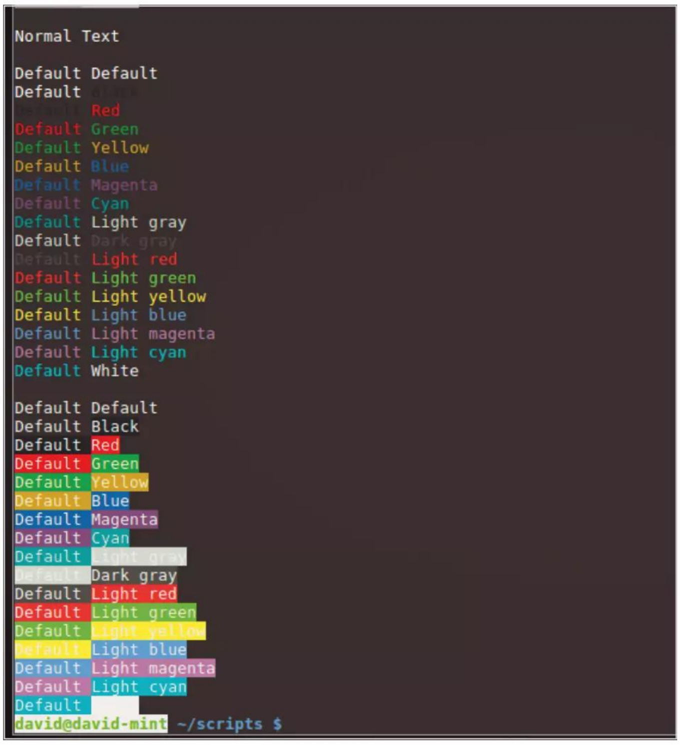
We briefly looked at putting some colours in the output for our scripts. Whilst it's too long to dig a little deeper into the colour options, here's a script that outputs what's available. Create a new script called `colours.sh` and enter the text (see below) into it.



```
colours.sh ✖
#!/bin/bash

clear
echo -e "Normal \e[1mBold"
echo -e "Normal \e[2mDim"
echo -e "Normal \e[4mUnderlined"
echo -e "Normal \e[5mBlink"
echo -e "Normal \e[7minverted"
echo -e "Normal \e[8mHidden"
echo
echo -e "\e[0mNormal Text"
echo

echo -e "Default \e[39mDefault"
echo -e "Default \e[30mBlack"
echo -e "Default \e[31mRed"
echo -e "Default \e[32mGreen"
echo -e "Default \e[33mYellow"
echo -e "Default \e[34mBlue"
echo -e "Default \e[35mMagenta"
echo -e "Default \e[36mCyan"
echo -e "Default \e[37mLight gray"
echo -e "Default \e[90mDark gray"
echo -e "Default \e[91mLight red"
echo -e "Default \e[92mLight green"
echo -e "Default \e[93mLight yellow"
echo -e "Default \e[94mLight blue"
echo -e "Default \e[95mLight magenta"
echo -e "Default \e[96mLight cyan"
echo -e "Default \e[97mWhite"
echo
echo -e "Default \e[49mDefault"
echo -e "Default \e[40mBlack"
echo -e "Default \e[41mRed"
echo -e "Default \e[42mGreen"
echo -e "Default \e[43mYellow"
echo -e "Default \e[44mBlue"
echo -e "Default \e[45mMagenta"
echo -e "Default \e[46mCyan"
echo -e "Default \e[47mLight gray"
echo -e "Default \e[100mDark gray"
echo -e "Default \e[101mLight red"
echo -e "Default \e[102mLight green"
echo -e "Default \e[103mLight yellow"
echo -e "Default \e[104mLight blue"
echo -e "Default \e[105mLight magenta"
echo -e "Default \e[106mLight cyan"
echo -e "Default \e[107mWhite"
```

STEP 8

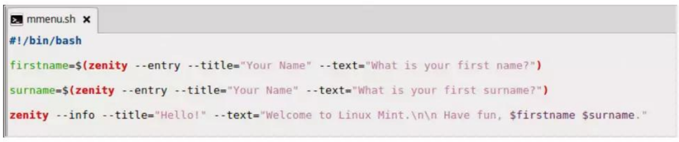
The output from colours.sh can, of course, be mixed together, bringing different effects depending on what you want to the output to say. For example, white text in a red background flashing (or blinking). Sadly the blinking effect doesn't work on all Terminals, so you may need to change to a different Terminal.



```
Normal Text

Default Default
Default Black
Default Red
Default Green
Default Yellow
Default Blue
Default Magenta
Default Cyan
Default Light gray
Default Dark gray
Default Light red
Default Light green
Default Light yellow
Default Light blue
Default Light magenta
Default Light cyan
Default White

Default Default
Default Black
Default Red
Default Green
Default Yellow
Default Blue
Default Magenta
Default Cyan
Default Light gray
Default Dark gray
Default Light red
Default Light green
Default Light yellow
Default Light blue
Default Light magenta
Default Light cyan
Default
david@david-mint ~/scripts $
```

STEP 9

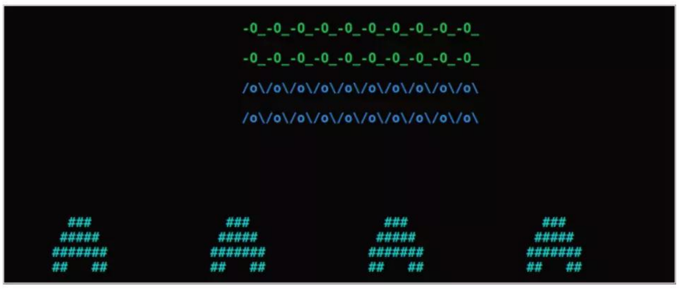
Whilst we're on making fancy scripts, how about using Zenity to output a graphical interface? Enter what you see below into a new script, `mmenu.sh`. Make it executable and then run it. You should have a couple of dialogue boxes appear, followed by a final message.



```
mmenu.sh ✖
#!/bin/bash

firstname=$(zenity --entry --title="Your Name" --text="What is your first name?")

surname=$(zenity --entry --title="Your Name" --text="What is your first surname?")

zenity --info --title="Hello!" --text="Welcome to Linux Mint.\n\n Have fun, $firstname $surname."
```

STEP 10

While gaming in a Bash script isn't something that's often touched upon, it is entirely possible, albeit, a little basic. Movement-based games are diffiuclt, and sometimes buggy, however a good text adventure, or Fighting Fantasy type game is a perfect choice for gaming in the Terminal. give it a go, and let us know how you get on.



```
                          -0_-0_-0_-0_-0_-0_-0_-0_-0_-0_

                          -0_-0_-0_-0_-0_-0_-0_-0_-0_-0_

                          /o\/o\/o\/o\/o\/o\/o\/o\/o\/o\

                          /o\/o\/o\/o\/o\/o\/o\/o\/o\/o\




    ###                 ###                 ###                 ###
   #####               #####               #####               #####
  #######             #######             #######             #######
  ##   ##             ##   ##             ##   ##             ##   ##
```

Command Line Quick Reference

When you start using Linux full time, you will quickly realise that the graphical interfaces of Ubuntu, Mint, etc. are great for many tasks but not great for all tasks. Understanding how to use the command line not only builds your understanding of Linux but also improves your knowledge of coding and programming in general. Our command line quick reference guide is designed to help you master Linux quicker.

TOP 10 COMMANDS

These may not be the most common commands used by everyone but they will certainly feature frequently for many users of Linux and the command line.

cd

The `cd` command is one of the commands you will use the most at the command line in Linux. It allows you to change your working directory. You use it to move around within the hierarchy of your file system. You can also use chdir.

ls

The `ls` command shows you the files in your current directory. Used with certain options, it lets you see file sizes, when files where created and file permissions. For example, `ls ~` shows you the files that are in your home directory.

cp

The `cp` command is used to make copies of files and directories. For example, `cp file sub` makes an exact copy of the file whose name you entered and names the copy sub but the first file will still exist with its original name.

pwd

The `pwd` command prints the full pathname of the current working directory (pwd stands for "print working directory"). Note that the GNOME terminal also displays this information in the title bar of its window.

clear

The `clear` command clears your screen if this is possible. It looks in the environment for the terminal type and then in the terminfo database to figure out how to clear the screen. This is equivalent to typing Control-L when using the bash shell.

mv

The `mv` command moves a file to a different location or renames a file. For example `mv file sub` renames the original file to `sub. mv sub ~/Desktop` moves the file 'sub' to your desktop directory but does not rename it. You must specify a new filename to rename a file.

chown

The `chown` command changes the user and/or group ownership of each given file. If only an owner (a user name or numeric user ID) is given, that user is made the owner of each given file, and the files' group is not changed.

cmod

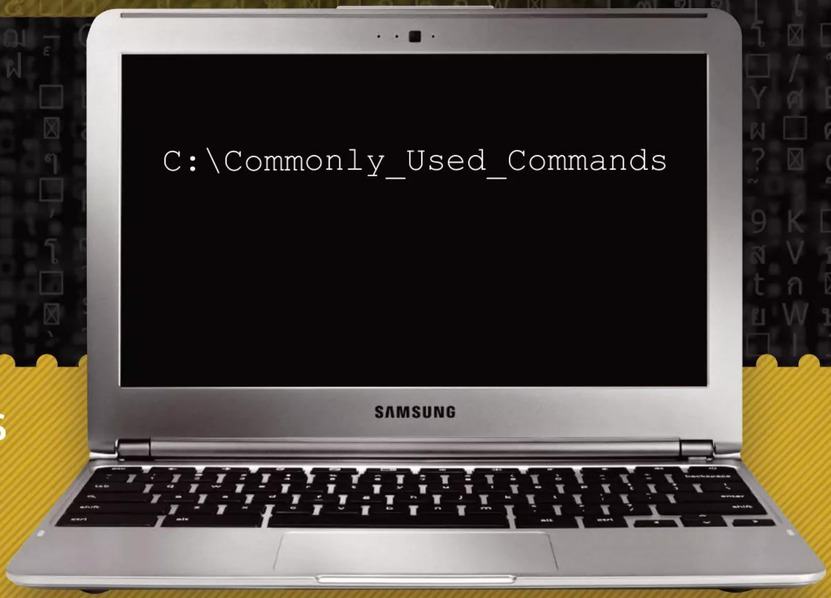
The `chmod` command changes the permissions on the files listed. Permissions are based on a fairly simple model. You can set permissions for user, group and world and you can set whether each can read, write and or execute the file.

rm

The `rm` command removes (deletes) files or directories. The removal process unlinks a filename in a filesystem from data on the storage device and marks that space as usable by future writes. In other words, removing files increases the amount of available space on your disk.

mkdir

Short for "make directory", `mkdir` is used to create directories on a file system, if the specified directory does not already exist. For example, `mkdir work` creates a work directory. More than one directory may be specified when calling mkdir.



USEFUL HELP/INFO COMMANDS

The following commands are useful for when you are trying to learn more about the system or program you are working with in Linux. You might not need them every day, but when you do, they will be invaluable.

free

The `free` command displays the total amount of free and used physical and swap memory in the system. For example, `free -m` gives the information using megabytes.

df

The `df` command displays filesystem disk space usage for all partitions. The command `df-h` is probably the most useful (the -h means human-readable).

top

The `top` program provides a dynamic real-time view of a running system. It can display system summary information, as well as a list of processes.

uname-a

The `uname` command with the `-a` option prints all system information, including machine name, kernel name, version and a few other details.

ps

The `ps` command allows you to view all the processes running on the machine. Every operating system's version of ps is slightly different but all do the same thing.

grep

The `grep` command allows you to search inside a number of files for a particular search pattern and then print matching lines. An example would be: `grep blah file`.

sed

The `sed` command opens a stream editor. A stream editor is used to perform text transformations on an input stream: a file or input from a pipeline.

adduser

The `adduser` command adds a new user to the system. Similarly, the `addgroup` command adds a new group to the system.

deluser

The `deluser` command removes a user from the system. To remove the user's files and home directory, you need to add the `-remove-home` option.

delgroup

The `delgroup` command removes a group from the system. You cannot remove a group that is the primary group of any users.

man man

The `man man` command brings up the manual entry for the man command, which is a great place to start when using it.

man intro

The `man intro` command is especially useful. It displays the Introduction to User Commands, which is a well written, fairly brief introduction to the Linux command line.

To help you on your way to becoming a better coder, we've included some great coding projects and ideas that you can apply to your own code, use as a demo for those who are starting to code, or improve and send out into the coding community. Want to learn how to craft some retro code? How about learning how to code a text adventure? These and more are all found in this final section.

No matter where you go next in coding, whether it's on to learning more languages, creating amazing apps and games, or helping out and teaching new coders to avoid the pitfalls you made, remember the basics, and have fun!

From here, it's down to you and your imagination. So dream it, and code it.

142 Planning your Code
144 Finding Resources
146 Creating a Loading Screen
148 Planning a Text Adventure
150 Text Adventure Script
152 Retro Coding
156 Common Coding Mistakes
158 Python Beginner's Mistakes
160 C++ Beginner's Mistakes

Code Projects and Ideas

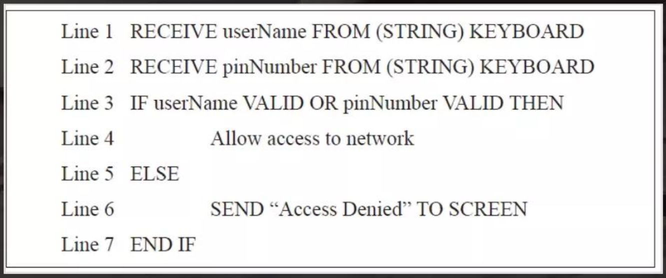
Planning Your Code

Knowing how to code is great, but it's not going to get you very far if you don't know what it is you're supposed to be coding. Small samples of code don't need much planning, but bigger projects do, so here are some tips on planning your code.

You will find that nearly all the professional developers out there will set aside some time to plan out how their code is going to fit into the wider project on which they're working. Regardless of whether they're working as part of a team on the latest game, developing a workflow user interface for the company, or creating a set of automated tasks that can be used across a number of different platforms, each developer will set out their code in a plan.

The planning stages can be unique to the developer, unique to the company for which they're working, or the project on which they're working. Planning meetings will often sketch out a rough plan of how the end product will function, with each developer team then working on their element within the project. Lone developers will sometimes go back to basics, grab paper and a pencil and start getting their ideas down. The end result is always the same, though, a plan of action that enables the developer to plan without worrying about the actual code at this early stage.

PSEUDOCODE



```
RECEIVE userName FROM (STRING) KEYBOARD
RECEIVE pinNumber FROM (STRING) KEYBOARD
IF userName VALID OR pinNumber VALID THEN
        Allow access to network
ELSE
        SEND "Access Denied" TO SCREEN
END IF
```

One of the more popular methods of planning to code is using pseudocode. It's terms used in programming circles that allows the developer to represent the implementation of code, the sequence of the actions, using plain English and a mixture of code examples together with a flowchart or two to help 'picture' loops within the coding project.

For example, a developer who is planning to write a program that will check user input for odd or even numbers will probably begin with a basic statement of pseudocode:

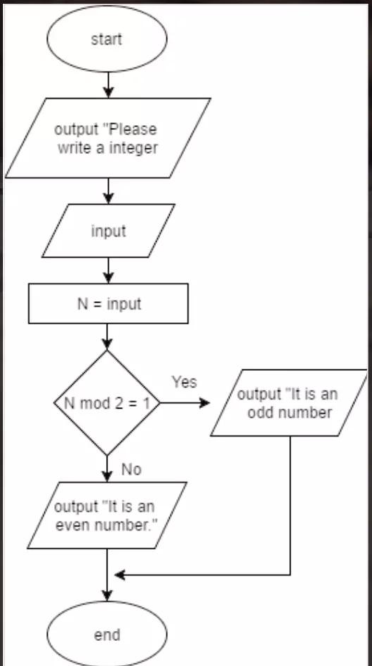
A program to allow a user to check if the number is odd or even. Then the basic statement can be expanded:

Code Example:

```
If "1"
        Print response
        "I am result 1"

If "2"
        Print response
        "I am result 2"
```

It's very simplistic, but from there the developer can continue to plan the remainder of the code needed without taking up too much valuable time – but while also having a readable plan of action for how the code is going to be formed.

The actual writing of the code, therefore, takes place later on in the project. Once there's a plan of action in place, the

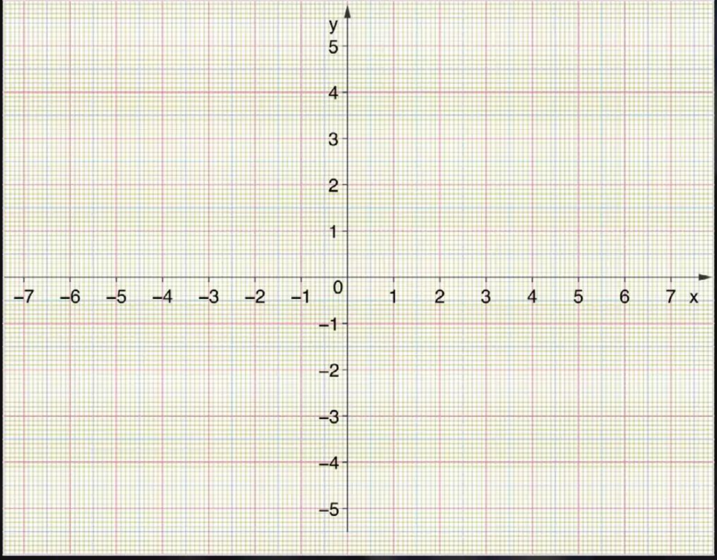

start
output "Please write a integer
input
N = input
N mod 2 = 1
Yes
output "It is an odd number
No
output "It is an even number."
end

developer can share the plan with their colleagues, lead developers, managers, and so on. Then the plan can be further tweaked to include other aspects that the original developer may have missed or may have been removed due to being being classed as unnecessary.

This process will not only trim the fat off the code before it's even written but will also help plug any gaps in the project before the hard work of entering the code begins.

GRAPH PAPER

If you're planning to include graphics in your project, such as a platform game, then one of the best resources you can use – and one that has been used since the early days of coding games – is graph paper.



Super Mario Bros., launched for the Nintendo Entertainment System in 1985, was nearly exclusively drawn out – both characters and levels – on graph paper before being applied to code. Entire game maps were often created this way, especially on the older 8-bit machines such as the Commodore 64 and Sinclair ZX Spectrum home computers. 8-Bit graphics were ideal for graph paper since the sprites used then were made up of eight or sixteen blocks; the developer could then visualise the layout of the sprite using the drawing on the graph paper and apply that to the code. The end result, of course, is the character appearing on the screen.

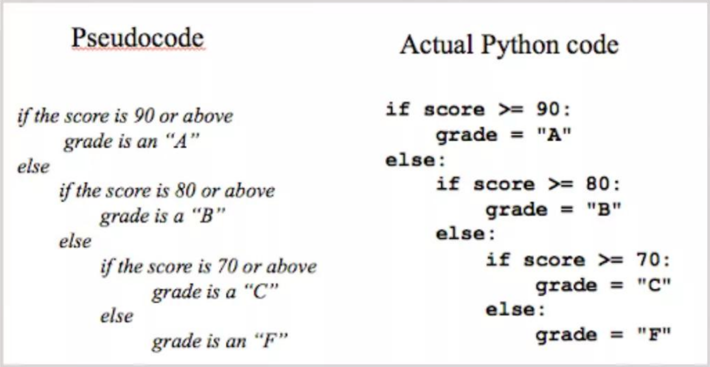
Once the graphical planning stage is complete, it's then down to you as the developer to fill in the blanks, as it were. This means creating the code that will place and animate your characters within the game, as well as developing interactions, collision detection, number of lives, and so on. With the help of some pseudocode, you could easily plan out what modules you'd use or create, the number of variables needed, and the various endgame states that will arise when someone plays your game.

GET PLANNING

The whole point of planning is to create something easy to follow that will help you write the code for your project. Planning isn't supposed to be a chore; it's there to aid you when you get stuck trying to work out which variables you've used, where you are in the game, with what you're left and how the game will eventually pan out and end.

When planning, begin with small steps. Coding a Hangman game, for example, begins with the plan to get the player's name, then includes the word bank, drawing out the stages of the Hangman, asking the right questions, and finishing the game in one of two states – a win or a lose – then asking if the player wants another go.

Once you're used to planning it'll come naturally, and you'll find yourself thankful for all those post-it notes stuck to a sheet of A3 paper, as well as the reams of graph paper that detail your characters and level designs.

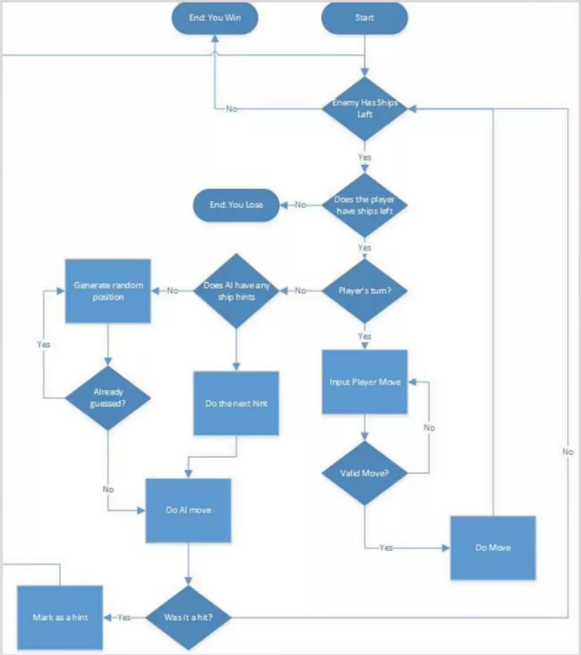


Pseudocode

```
if the score is 90 or above
    grade is an "A"
else
    if the score is 80 or above
        grade is a "B"
    else
        if the score is 70 or above
            grade is a "C"
        else
            grade is an "F"
```

Actual Python code

```
if score >= 90:
    grade = "A"
else:
    if score >= 80:
        grade = "B"
    else:
        if score >= 70:
            grade = "C"
        else:
            grade = "F"
```



Pseudocode flowcharts can be simple, or as complex as necessary

Writing pseudocode will help you visualise the actual code

Finding Resources

The Python community is huge and spans countless websites and forums on the Internet, so finding extra help here and there isn't too much of a problem. However, finding good Python resources can be, so here are some ideas for you to try.

You will reach a point in your Python coding experience when you need to rely on some external resources to help you through a sticky patch in your project or give you a bit of inspiration. Python resources are available throughout the Internet, but not all of them are any good. If you spend any time looking up code for a particular project, then you're likely to come across copious examples, yet not all of them seem to work.

That's one of the main issues with resources for Python. With the language being twenty-nine years old, and many different versions under the bridge, there's a significant amount of content out there that's simply out of date – or just wrong for the version you're using. Newer versions, even by one number out, can have different outputs for your code from a previous version – it's not always that case, but it can happen. Therefore, you need to find yourself some trusted Python resources that work, or at least 90% would work, with the version you're using and the code within.

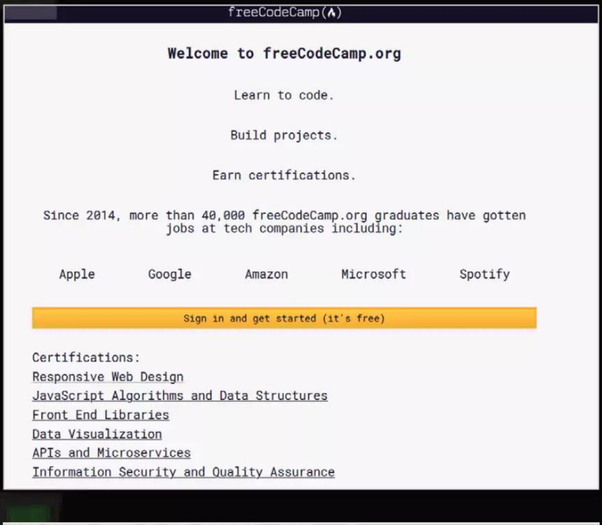
THE HITCHHIKER'S GUIDE TO PYTHON

One of the first places that's worth turning to is: freeCodeCamp.org. This website features a wealth of code that covers most aspects of the learning experience, even for those who are experts at Python. You'll find content that will take your Python journey to new heights; with code snippets to help you access networking elements, military-style encryption and more, and you can even gain certification through the site.

Programiz is another useful resource that features Python code examples covering a wide range of concepts. There's everything in here from the original "Hello, world!" code, through to code that will transpose a matrix or find the hash of a file. There's also an online compiler and in-depth looks into further reading for aspiring data scientists.

If you're looking for resources about using graphics in Python, then ProgramArcadeGames.com is a good place to start. This site has plenty of code for the Pygame module, including moving an object around the screen using a mouse or game controller, creating fractals, moving sprites, and an example of a classic Pong game for you to try out.

GitHub is always worth a look. However, the code featured here isn't always up to date, and you may find yourself having to convert Python 2 code into Python 3. Nevertheless, you do come across some gems from time to time, especially if you look up the Pygame examples that users have uploaded. We recommend the excellent Solarwolf, by Pete Shinners.



Welcome to freeCodeCamp.org

Learn to code.

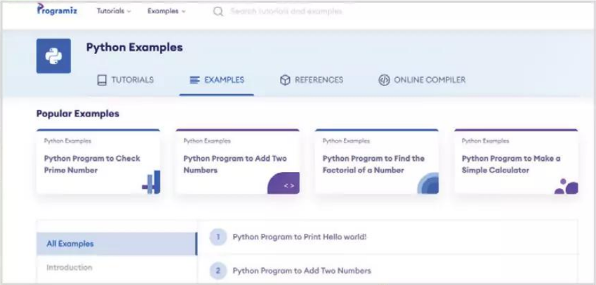
Build projects.

Earn certifications.

Since 2014, more than 40,000 freeCodeCamp.org graduates have gotten jobs at tech companies including:

Apple Google Amazon Microsoft Spotify

Certifications:
Responsive Web Design
JavaScript Algorithms and Data Structures
Front End Libraries
Data Visualization
APIs and Microservices
Information Security and Quality Assurance



Python Examples

Popular Examples

Python Examples: Python Program to Check Prime Number

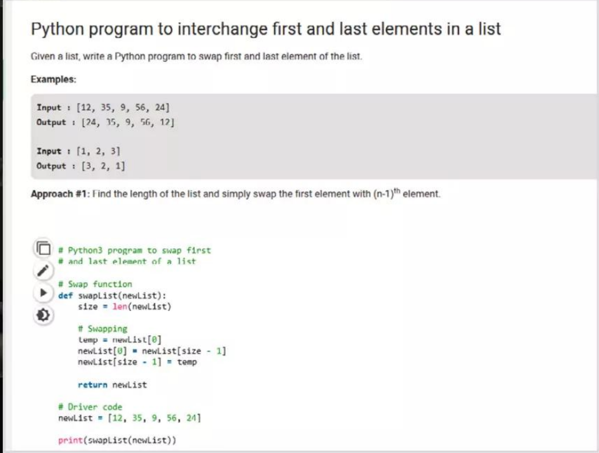
Python Examples: Python Program to Add Two Numbers

Python Examples: Python Program to Find the Factorial of a Number

Python Examples: Python Program to Make a Simple Calculator

1 Python Program to Print Hello world!

2 Python Program to Add Two Numbers



Python program to interchange first and last elements in a list

Given a list, write a Python program to swap first and last element of the list.

Examples:

```
Input : [12, 35, 9, 56, 24]
Output : [24, 35, 9, 56, 12]

Input : [1, 2, 3]
Output : [3, 2, 1]
```

Approach #1: Find the length of the list and simply swap the first element with (n-1)th element.

```
# Python3 program to swap first
# and last element of a list

# Swap function
def swapList(newList):
    size = len(newList)

    # Swapping
    temp = newList[0]
    newList[0] = newList[size - 1]
    newList[size - 1] = temp

    return newList

# Driver code
newList = [12, 35, 9, 56, 24]

print(swapList(newList))
```

Guardian Info

The Guardians protect the Power Cubes on every level. They will become more aggressive in the later levels.

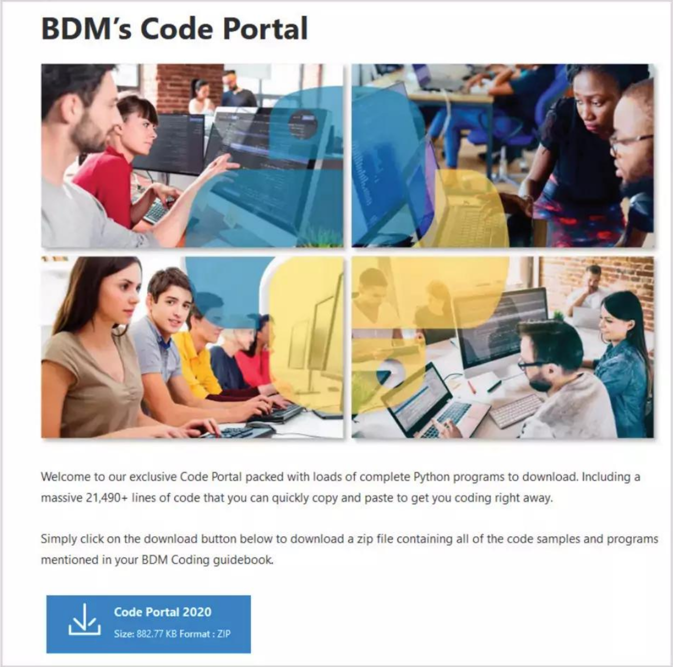
Don't get too close, they like to shoot point blank.

STRONG GOOGLE-FU

There's no trick to finding good Python resources online; most of the time, it's a case of trial and error. You'll need to make sure you're searching for the right content, though, just to save you some time. If it's a Pygame specific set of resources you're looking for then you'll need to specify the Python version you're using; include Pygame, and include what it is you're looking for – it may sound like we're teaching you to suck eggs, but effective searching is often overlooked.

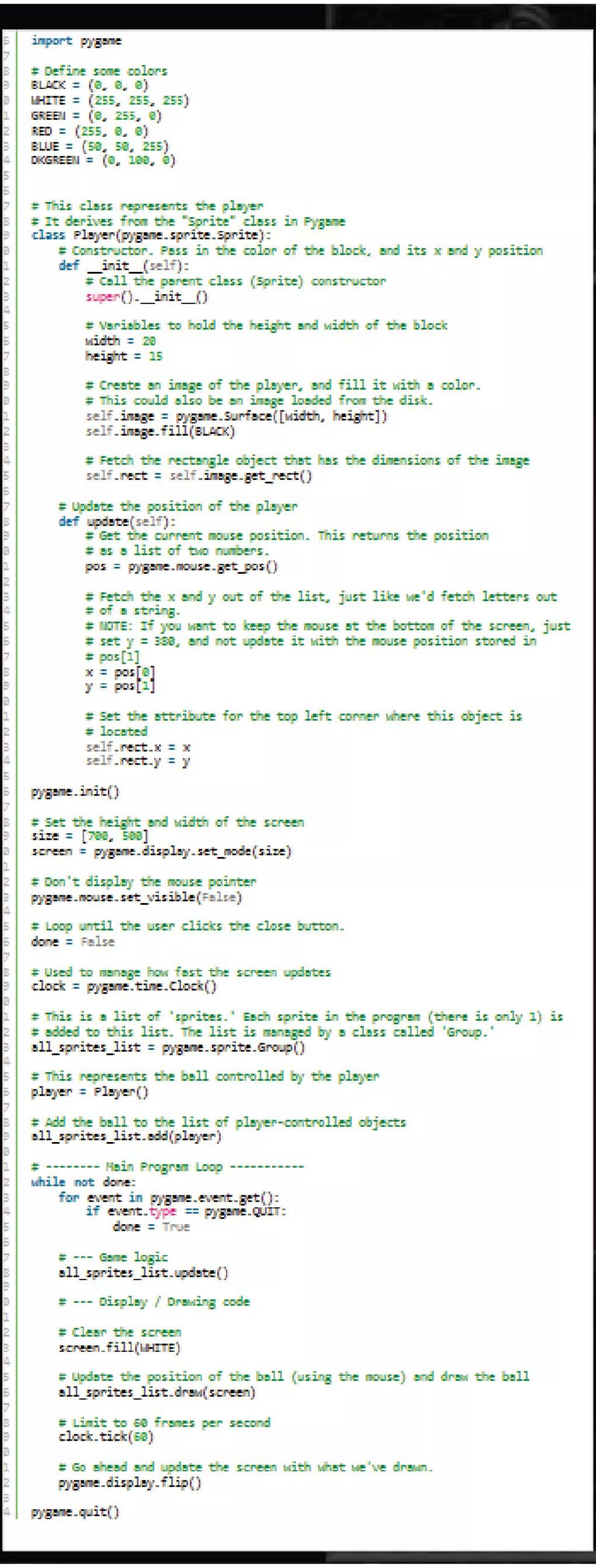
Again, it's also worth looking up working examples of a type of game if you're looking for Pygame content. If you want to learn how to engineer sprites moving from one side of the screen to the other, then look up some examples of Space Invaders, which feature such movements. The same goes if you're making a platform game, examine some classic games that involved platforming, like Mario and so on.

One last resource is our very own code portal. Log in to **https://bdmpublications.com/code-portal/**, and you'll find a continually growing list of Python code and resources that you can freely add to your own code, then chop, change, and use for whatever projects you have in mind.


BDM's Code Portal

Welcome to our exclusive Code Portal packed with loads of complete Python programs to download. Including a massive 21,490+ lines of code that you can quickly copy and paste to get you coding right away.

Simply click on the download button below to download a zip file containing all of the code samples and programs mentioned in your BDM Coding guidebook.


```
import pygame

# Define some colors
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLUE = (50, 50, 255)
DKGREEN = (0, 100, 0)


# This class represents the player
# It derives from the "Sprite" class in Pygame
class Player(pygame.sprite.Sprite):
    # Constructor. Pass in the color of the block, and its x and y position
    def __init__(self):
        # Call the parent class (Sprite) constructor
        super().__init__()

        # Variables to hold the height and width of the block
        width = 20
        height = 15

        # Create an image of the player, and fill it with a color.
        # This could also be an image loaded from the disk.
        self.image = pygame.Surface([width, height])
        self.image.fill(BLACK)

        # Fetch the rectangle object that has the dimensions of the image
        self.rect = self.image.get_rect()

    # Update the position of the player
    def update(self):
        # Get the current mouse position. This returns the position
        # as a list of two numbers.
        pos = pygame.mouse.get_pos()

        # Fetch the x and y out of the list, just like we'd fetch letters out
        # of a string.
        # NOTE: If you want to keep the mouse at the bottom of the screen, just
        # set y = 380, and not update it with the mouse position stored in
        # pos[1]
        x = pos[0]
        y = pos[1]

        # Set the attribute for the top left corner where this object is
        # located
        self.rect.x = x
        self.rect.y = y

pygame.init()

# Set the height and width of the screen
size = [700, 500]
screen = pygame.display.set_mode(size)

# Don't display the mouse pointer
pygame.mouse.set_visible(False)

# Loop until the user clicks the close button.
done = False

# Used to manage how fast the screen updates
clock = pygame.time.Clock()

# This is a list of 'sprites.' Each sprite in the program (there is only 1) is
# added to this list. The list is managed by a class called 'Group.'
all_sprites_list = pygame.sprite.Group()

# This represents the ball controlled by the player
player = Player()

# Add the ball to the list of player-controlled objects
all_sprites_list.add(player)

# -------- Main Program Loop -----------
while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True

    # --- Game logic
    all_sprites_list.update()

    # --- Display / Drawing code

    # Clear the screen
    screen.fill(WHITE)

    # Update the position of the ball (using the mouse) and draw the ball
    all_sprites_list.draw(screen)

    # Limit to 60 frames per second
    clock.tick(60)

    # Go ahead and update the screen with what we've drawn.
    pygame.display.flip()

pygame.quit()
```

Creating a Loading Screen

If you're looking to add a little something extra to your Python code, then consider including a loading screen. The loading screen is a short introduction, or piece of art, that appears before the main part of your code.

3D STARSTRIKE
© REALTIME SOFTWARE 84

LOAD""

Back in the 80s, in the 8-bit home computing era, loading screens were often used to display the cover of a game as it loaded from tape. The image would load itself in, usually one-line-at-a-time, then proceed to colour itself in while the loading raster bars danced around in the borders of the screen.

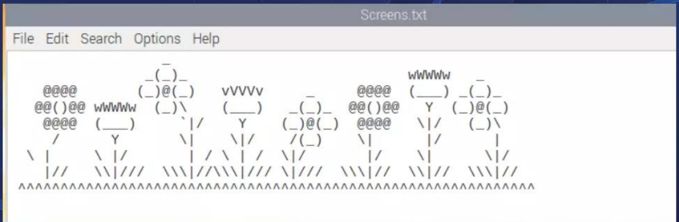
Loading screens were a part of the package and often the buy-in for the whole game as an experience. Some loading screens featured animations, or a countdown for time remaining as the game loads, while others even went so far as to include some kind of playable game. The point being: a loading screen is not just an artistic part of computing history, but an introduction to the program that's about to be run.

While these days loading screens may no longer be with us, in terms of modern gaming we can still include them in our own Python content. Either just for fun, or to add a little retro-themed spice to the mix.

SCREEN$

Creating a loading screen in Python is remarkably easy. You have several options to hand: you can create a tkinter window and display an image, followed by a brief pause, before starting your main code, or you could opt for a console-based ASCII art version that loads one-line-at-a-time. Let's look at the latter and see how it works.

First you'll need some ASCII art, you can look up plenty of examples online, or use an image to ASCII Art converter to transform any images you have to ASCII. When you have your ASCII art, drop it into a newly created folder inside a normal text file.

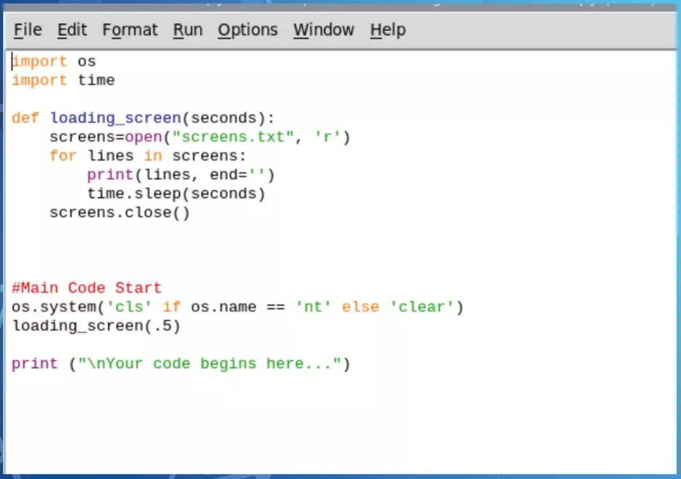


```
                 _
               _(_)_                          wWWWWw  _
   @@@@       (_)@(_)   vVVVv     _     @@@@  (___) _(_)_
  @@()@@ wWWWWw (_)\    (___)   _(_)_  @@()@@   Y  (_)@(_)
   @@@@  (___)     `|/    Y    (_)@(_)  @@@@   \|/   (_)\
    /      Y       \|    \|/    /(_)    \|      |/      |
 \ |     \ |/       | / \ | /  \|/       |/    \|      \|/
   |//   \\|///  \\\|//\\\|/// \|///  \\\|//  \\|//  \\\|//
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

THE CODE

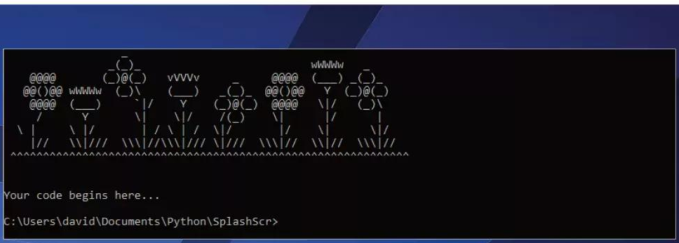
Launch Python and enter the following code to a New File:

```
import os
import time

def loading_screen(seconds):
    screens=open(“screens.txt”, ‘r’)
    for lines in screens:
        print(lines, end=’’)
        time.sleep(seconds)
    screens.close()

#Main Code Start
os.system(‘cls’ if os.name == ‘nt’ else
‘clear’)
loading_screen(.5)

print (“\nYour code begins here...”)
```



```
import os
import time

def loading_screen(seconds):
    screens=open("screens.txt", 'r')
    for lines in screens:
        print(lines, end='')
        time.sleep(seconds)
    screens.close()



#Main Code Start
os.system('cls' if os.name == 'nt' else 'clear')
loading_screen(.5)

print ("\nYour code begins here...")
```

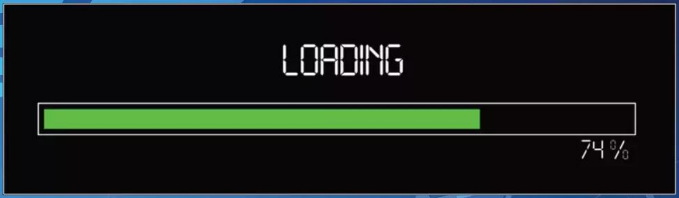
The code is quite simple: import the OS and Time modules and then create a Python function called loading_screen with a (seconds) option. Within the function, open the text file with the ASCII art as read-only and create a `For` loop that'll read the text file one-line-at-a-time. Next, print the lines – incidentally, the `lines, end=’’` element will strip the newline from the text document, without it you'll end up with a double-line spaced display. Include the timing in seconds and close the text file buffer. The final part of the code, `#Main Code Start`, is where you'll clear the screen (CLS for Windows, Clear for Linux (Raspberry Pi) and macOS) and call the function, together with the output number of seconds to wait for each line to be written to the screen – in this case, .5 of a second.



```
Your code begins here...

C:\Users\david\Documents\Python\SplashScr>
```

Save the code as screens.py, drop into a Command Prompt or Terminal and execute it. The screen will clear and your ASCII art inside the text file will load in line-by-line, creating a loading screen effect.

LOADING...



Another favourite introduction screen is that of a simple loading animation, where the word loading is displayed, followed by some characters, and a percentage of the program loaded. While it may not take long for your Python code to load, the effect can be interesting.

Create a New File in Python and enter the following code:

```
import os
import time

def loading_bar(seconds):
  for loading in range(0,seconds+1):
    percent = (loading * 100) // seconds
    print(“\n”)
    print(“Loading...”)
    print(“<” + (“-” * loading) + (“ “ * (seconds-
loading)) + “> “ + str(percent) + “%”)
    print(“\n”)
    time.sleep(1)
    os.system(‘cls’ if os.name == ‘nt’ else
‘clear’)

#Main Code Start
loading_bar(10)

print (“\nYour code begins here...”)
```

The code works in much the same way as the previous, except, instead of reading from a text file, it's running through a For loop that prints Loading… followed by an animation of sorts, along with a percentage counter; clearing the screen every second and displaying the new results. It's simple, yes, but quite effective in its design.

COMBINING THE TWO

How about combining the two elements we've looked at? Let's begin with a Loading… progress bar, followed by the loading screen. After that, you can include your own code and continue your program. Here's the code:

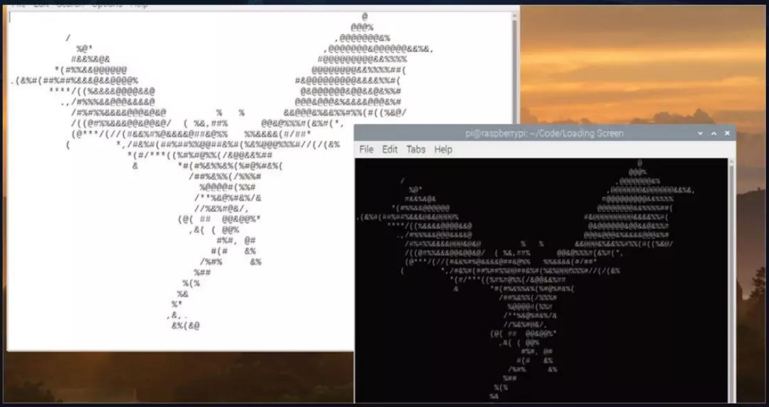
```
import os
import time

def loading_bar(seconds):
  for loading in range(0,seconds+1):
    percent = (loading * 100) // seconds
    print(“\n”)
    print(“Loading...”)
    print(“<” + (“-” * loading) + (“ “ * (seconds-
loading)) + “> “ + str(percent) + “%”)
    print(“\n”)
    time.sleep(1)
    os.system(‘cls’ if os.name == ‘nt’ else
‘clear’)

def loading_screen(seconds):
    screens=open(“screens.txt”, ‘r’)
    for lines in screens:
        print(lines, end=’’)
        time.sleep(seconds)
    screens.close()

#Main Code Start
loading_bar(10)
os.system(‘cls’ if os.name == ‘nt’ else ‘clear’)
loading_screen(.5)

print (“\nYour code begins here...”)
```

You can, of course, pull those functions up wherever and whenever you want in your code and they'll display, as they should, at the beginning. Remember to have the ASCII text file in the same folder as the Python code, or, at the screens=open("screens.txt", 'r') part of the code, point to where the text file is located.

ADVENTURE TIME

A good example of using loading screen, ASCII art text images is when coding a text adventure. Once you've established your story, created the characters, events and so on, you could easily incorporate some excellently designed ASCII art to your game.

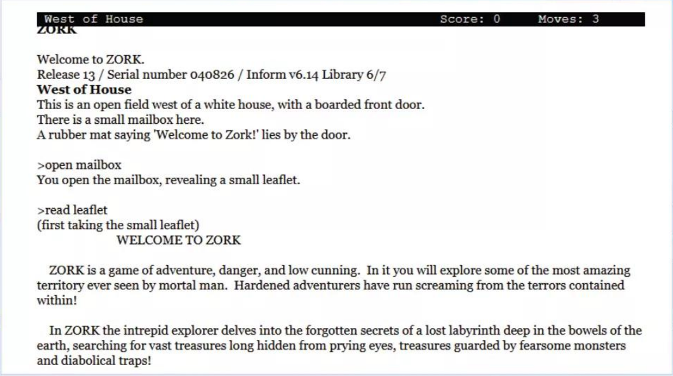
Imagine coming across a dragon, in game, and displaying its representation as ASCII. You can then load up the image lines, one-by-one, and continue with the rest of the adventure code. It's certainly worth having a play around with and it'll definitely add a little something else extra.



Planning a Text Adventure

Planning a text adventure is one of the best ways of teaching yourself some new Python tricks and essential code. A good text adventure contains many aspects, all of which will need to be carefully planned and structured, to get the best from the code.

If you can think back to some of the classic text adventures you may have played in the past, consider the amount of work that was needed to get them up and running. If you're not familiar with a text adventure, then open a browser and go to **http://textadventures.co.uk/games/play/5zyoqrsugeopel3ffhz_vq**. This is an online version of the classic text adventure Zork. Zork was created for the DEC PDP-10 workstations, using the MDL programming language back in the late seventies. Despite its age, it's still considered one of the best text adventures to play.

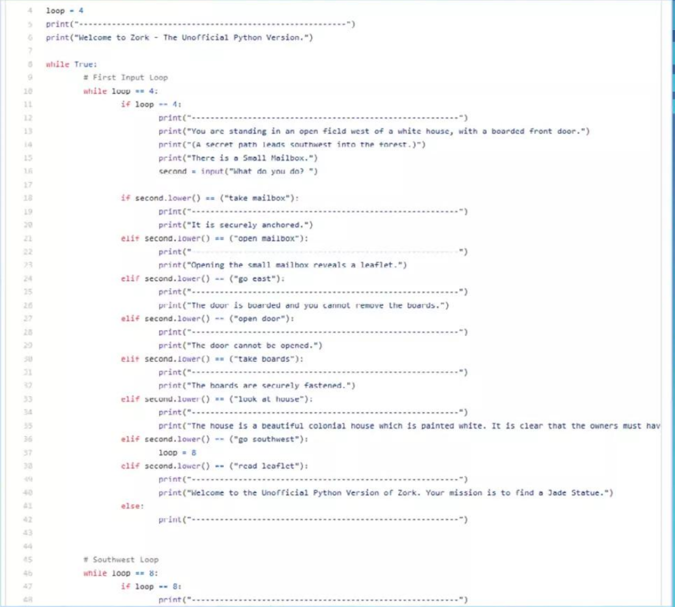


```
 West of House                              Score: 0      Moves: 3
ZORK

Welcome to ZORK.
Release 13 / Serial number 040826 / Inform v6.14 Library 6/7
West of House
This is an open field west of a white house, with a boarded front door.
There is a small mailbox here.
A rubber mat saying 'Welcome to Zork!' lies by the door.

>open mailbox
You open the mailbox, revealing a small leaflet.

>read leaflet
(first taking the small leaflet)
                WELCOME TO ZORK

  ZORK is a game of adventure, danger, and low cunning.  In it you will explore some of the most amazing
territory ever seen by mortal man.  Hardened adventurers have run screaming from the terrors contained
within!

  In ZORK the intrepid explorer delves into the forgotten secrets of a lost labyrinth deep in the bowels of the
earth, searching for vast treasures long hidden from prying eyes, treasures guarded by fearsome monsters
and diabolical traps!
```

Zork, or any other text adventure game, features several elements necessary to make the game both interesting and playable.

INGREDIENTS FOR A COMPELLING ADVENTURE

The first ingredient is a story, a plot that will take the player on a ride through their own imagination. It doesn't necessarily have to include dragons, barbarians, spaceships, or other such things, but it needs to be engaging. Perhaps it could follow the theme of a murder mystery, much in the same vein as an Agatha Christie novel, or it could be something that explores the base fears of people.



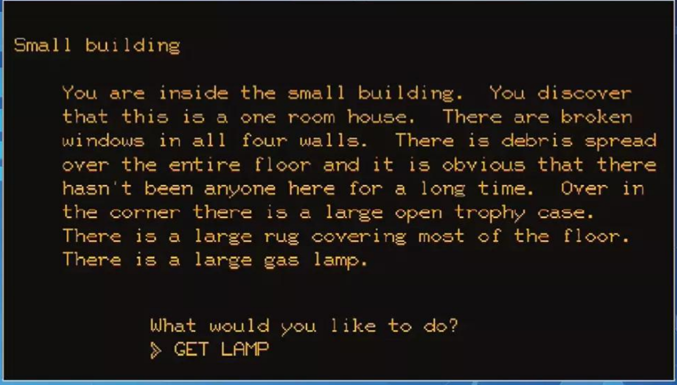
```python
loop = 4
print("------------------------------------------------------------")
print("Welcome to Zork - The Unofficial Python Version.")

while True:
    # First Input Loop
    while loop == 4:
        if loop == 4:
            print("------------------------------------------------------------")
            print("You are standing in an open field west of a white house, with a boarded front door.")
            print("(A secret path leads southwest into the forest.)")
            print("There is a Small Mailbox.")
            second = input("What do you do? ")

        if second.lower() == ("take mailbox"):
            print("------------------------------------------------------------")
            print("It is securely anchored.")
        elif second.lower() == ("open mailbox"):
            print("------------------------------------------------------------")
            print("Opening the small mailbox reveals a leaflet.")
        elif second.lower() == ("go east"):
            print("------------------------------------------------------------")
            print("The door is boarded and you cannot remove the boards.")
        elif second.lower() == ("open door"):
            print("------------------------------------------------------------")
            print("The door cannot be opened.")
        elif second.lower() == ("take boards"):
            print("------------------------------------------------------------")
            print("The boards are securely fastened.")
        elif second.lower() == ("look at house"):
            print("------------------------------------------------------------")
            print("The house is a beautiful colonial house which is painted white. It is clear that the owners must hav
        elif second.lower() == ("go southwest"):
            loop = 8
        elif second.lower() == ("read leaflet"):
            print("------------------------------------------------------------")
            print("Welcome to the Unofficial Python Version of Zork. Your mission is to find a Jade Statue.")
        else:
            print("------------------------------------------------------------")


    # Southwest Loop
    while loop == 8:
        if loop == 8:
            print("------------------------------------------------------------")
```

The second crucial element is choice. Without that, the player may as well read the story from a book. It's the choice presented to the player that makes the game an adventure. If the player chooses something wrong, they could be punished, whereas a correct choice could gain them gold or some other form of reward. However, it doesn't always need to be a positive or negative choice; after all, most common real-life decisions we make aren't life or death, they merely result in one thing happening or another.

Locations are similarly important. Text Adventures rely on good the locations featured in the game can draw a picture in the mind of the player. Whether they're on a wooden ship in the middle of the sea or trying to escape a haunted mansion, each location has to stand out.

Items are also a vital requirement for a good adventure game. A player will need to be able to pick up, examine, take, and use, an item from its location in the game. For example, your character could find a key in the first location, pocket it, and then use it later on in the game to unlock a door. You'll need to factor in the list of possible items, and you'll also need to factor in the real-world physics of each item too. If you have your adventurer enter a room with an anvil, then allowing them to pick up the anvil and put it in their pocket isn't particularly realistic. One of the major drawbacks of some text adventures, and even modern graphical games, is the unrealistic carrying capability of the player.

Although not always necessary, puzzles do form an interesting aspect of a text adventure. These puzzles can test the player's abilities and expand the game beyond simply finding an object and using it. Puzzles can range from using these objects correctly to achieve something to finding the correct route through a maze, with some being vital to the plot while others are simply there to give the player something interesting to do. Ideally, puzzles should be easy near the beginning of the game and get progressively harder as your character advances through the game.



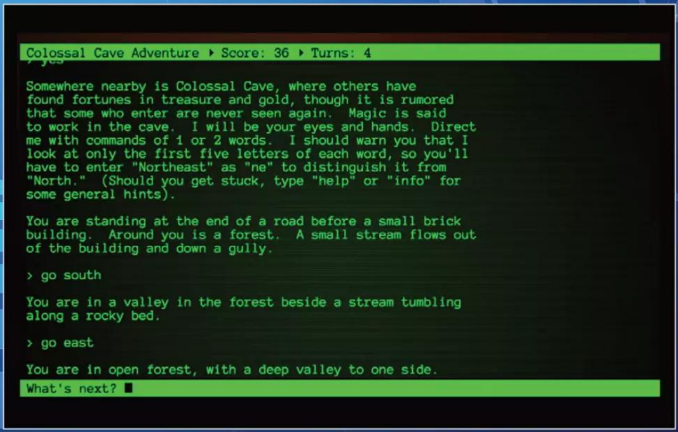
```
Small building

    You are inside the small building.  You discover
    that this is a one room house.  There are broken
    windows in all four walls.  There is debris spread
    over the entire floor and it is obvious that there
    hasn't been anyone here for a long time.  Over in
    the corner there is a large open trophy case.
    There is a large rug covering most of the floor.
    There is a large gas lamp.


           What would you like to do?
           > GET LAMP
```

Hints and help along the way can make the adventure seem less daunting. Of course, if you're particularly mean-spirited, then you can opt to exclude any form of help along the way altogether. However, most text adventures do include some form of a hint. If you've created a great set of puzzles for the player to beat, and they're stuck in their progression of the game, then allowing them to enter the word: help, or hint, into the system, will reveal a small clue as to what to do next. It's not cheating, just keeping the story moving along.

Pacing is the final aspect of the adventure game that needs to be planned correctly. Good pacing within the story is vital to keeping your player's attention, as well as keeping them entertained and engaged with what's going on. It's not an easy egg to crack, however. Too much pace and the player can get lost. Too slow, and they'll get bored and move on to something else.

THE CODE

It's impossible to list the code you'll need for your adventure game since it'll be different from each reader to the next, but here are some pointers to help you out.



```
Colossal Cave Adventure ▸ Score: 36 ▸ Turns: 4

Somewhere nearby is Colossal Cave, where others have
found fortunes in treasure and gold, though it is rumored
that some who enter are never seen again.  Magic is said
to work in the cave.  I will be your eyes and hands.  Direct
me with commands of 1 or 2 words.  I should warn you that I
look at only the first five letters of each word, so you'll
have to enter "Northeast" as "ne" to distinguish it from
"North."  (Should you get stuck, type "help" or "info" for
some general hints).

You are standing at the end of a road before a small brick
building.  Around you is a forest.  A small stream flows out
of the building and down a gully.

> go south

You are in a valley in the forest beside a stream tumbling
along a rocky bed.

> go east

You are in open forest, with a deep valley to one side.
What's next? █
```

Lists, tuples and dictionaries will be needed to store the various elements of the game. These can be descriptions of characters whom you'll meet, either friend or foe. Locations, with descriptions, throughout the game; items you'll come across that can be used, along with a player inventory, which will grow or lessen depending on what the player has used/dropped/picked up.

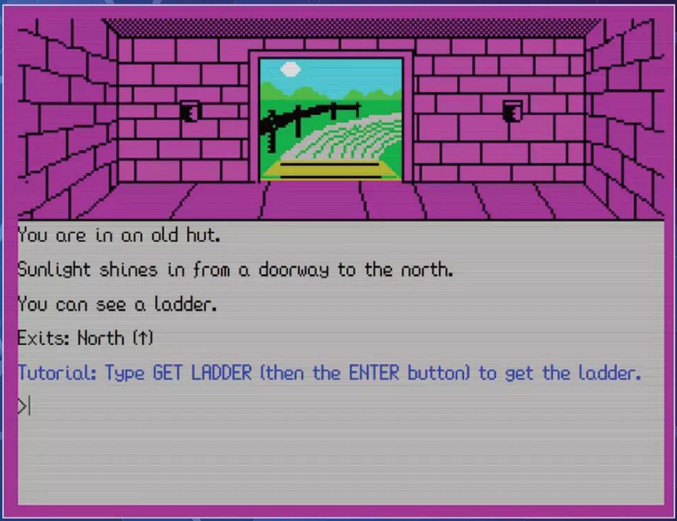
Variables will be needed to hold vital statistics during gameplay, the player's name, amount of strength, endurance, skill and health, for example. You'll also need variables to hold the current score or number of moves used so far, plus any statistics regarding the other characters you'll meet.

Randomly generated numbers are vital for those who want to include an element of combat. Within such examples, a virtual dice roll will determine the winner or loser, either in a fight or when using luck to win through something.

Strings can be used to keep non-moving elements of the game, in other words the names of the rooms you may enter. These components are static throughout the game, each and every time you play, so they can be called upon when the player interacts with them.

Functions are the lifeblood of a text adventure, as they will form the backbone of the locations, interactions, events and anything else that happens within the game.

Loops will keep a process going, such as a fight with an orc, until an event occurs – either the death of the orc or your player. In addition, you'll need to include a loop, if and else components to help out with choices made: do you take the green door or the red door?



```
You are in an old hut.
Sunlight shines in from a doorway to the north.
You can see a ladder.
Exits: North (↑)
Tutorial: Type GET LADDER (then the ENTER button) to get the ladder.
>|
```

GOOD LUCK, ADVENTURER

There's a lot to take in and plan, so our advice is to start small and plan out your story first. Once you've got the basic story, build on the settings and locations, then puzzles and other elements. From there, you can begin to expand the game world and add things like combat.

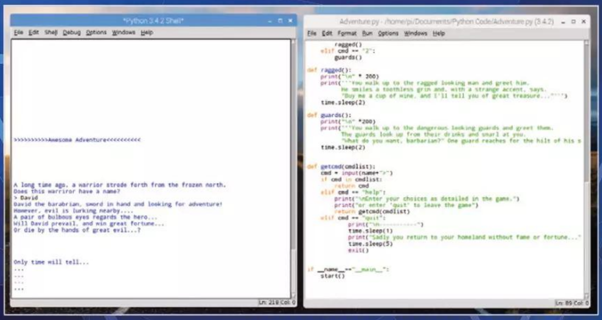
It won't be easy, but it'll be great fun and will help you become a better Python coder.

Text Adventure Script

Text adventures are an excellent way to build your Python coding skills and have some fun at the same time. This example that we created will start you on the path to making a classic text adventure; where it will end is up to you.

ADVENTURE.PY

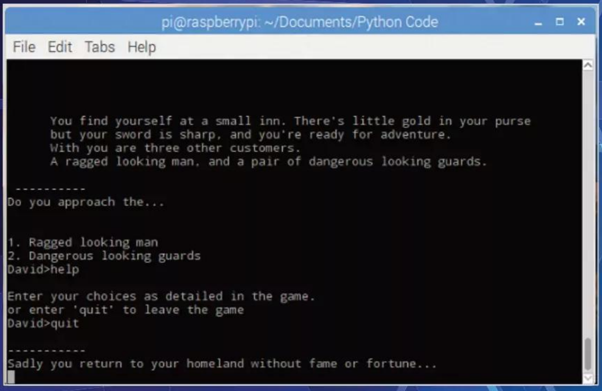
The Adventure game uses just the Time module to begin with, creating pauses between print functions. There's a help system in place to expand upon, as well as the story itself.



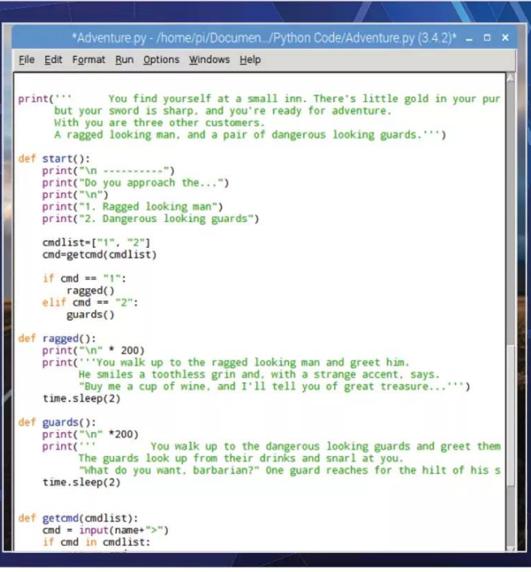
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Ln: 218 Col: 0
Adventure.py - /home/pi/Documents/Python Code/Adventure.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 89 Col: 0

```
>>>>>>>>>>Awesome Adventure<<<<<<<<<<



A long time ago, a warrior strode forth from the frozen north.
Does this warrior have a name?
> David
David the barabrian, sword in hand and looking for adventure!
However, evil is lurking nearby....
A pair of bulbous eyes regards the hero...
Will David prevail, and win great fortune...
Or die by the hands of great evil...?



Only time will tell...
...
...
...
...
```

```
        ragged()
    elif cmd == "2":
        guards()

def ragged():
    print("\n" * 200)
    print('''You walk up to the ragged looking man and greet him.
          He smiles a toothless grin and, with a strange accent, says.
          "Buy me a cup of wine, and I'll tell you of great treasure...''')
    time.sleep(2)

def guards():
    print("\n" *200)
    print('''You walk up to the dangerous looking guards and greet them.
          The guards look up from their drinks and snarl at you.
          "What do you want, barbarian?" One guard reaches for the hilt of his s
    time.sleep(2)


def getcmd(cmdlist):
    cmd = input(name+">")
    if cmd in cmdlist:
        return cmd
    elif cmd == "help":
        print("\nEnter your choices as detailed in the game.")
        print("or enter 'quit' to leave the game")
        return getcmd(cmdlist)
    elif cmd == "quit":
            print("\n-----------")
            time.sleep(1)
            print("Sadly you return to your homeland without fame or fortune..."
            time.sleep(5)
            exit()


if __name__=="__main__":
    start()
```



```


    You find yourself at a small inn. There's little gold in your purse
    but your sword is sharp, and you're ready for adventure.
    With you are three other customers.
    A ragged looking man, and a pair of dangerous looking guards.

 ----------
Do you approach the...


1. Ragged looking man
2. Dangerous looking guards
David>help

Enter your choices as detailed in the game.
or enter 'quit' to leave the game
David>quit

-----------
Sadly you return to your homeland without fame or fortune...
```



Adventure.py - /home/pi/Documen.../Python Code/Adventure.py (3.4.2)
File Edit Format Run Options Windows Help

```
print('''      You find yourself at a small inn. There's little gold in your pur
      but your sword is sharp, and you're ready for adventure.
      With you are three other customers.
      A ragged looking man, and a pair of dangerous looking guards.''')

def start():
    print("\n ---------")
    print("Do you approach the...")
    print("\n")
    print("1. Ragged looking man")
    print("2. Dangerous looking guards")

    cmdlist=["1", "2"]
    cmd=getcmd(cmdlist)

    if cmd == "1":
        ragged()
    elif cmd == "2":
        guards()

def ragged():
    print("\n" * 200)
    print('''You walk up to the ragged looking man and greet him.
          He smiles a toothless grin and, with a strange accent, says.
          "Buy me a cup of wine, and I'll tell you of great treasure...''')
    time.sleep(2)

def guards():
    print("\n" *200)
    print('''         You walk up to the dangerous looking guards and greet them
          The guards look up from their drinks and snarl at you.
          "What do you want, barbarian?" One guard reaches for the hilt of his s
    time.sleep(2)


def getcmd(cmdlist):
    cmd = input(name+">")
    if cmd in cmdlist:
```

```

import time

print("\n" * 200)
print(">>>>>>>>>>Awesome Adventure<<<<<<<<<<\n")
print("\n" * 3)
time.sleep(3)
print("\nA long time ago, a warrior strode forth from
the frozen north.")
time.sleep(1)
print("Does this warrior have a name?")
name=input("> ")
print(name, "the barbarian, sword in hand and looking
for adventure!")
time.sleep(1)
print("However, evil is lurking nearby....")
time.sleep(1)
print("A pair of bulbous eyes regards the hero...")
time.sleep(1)
print("Will", name, "prevail, and win great fortune...")
time.sleep(1)
print("Or die by the hands of great evil...?")
time.sleep(1)
print("\n" *3)
print("Only time will tell...")
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(1)
print('...')
time.sleep(5)
print("\n" *200)

print('''      You find yourself at a small inn. There's
   little gold in your purse but your sword is sharp,
   and you're ready for adventure.
   With you are three other customers.
   A ragged looking man, and a pair of dangerous
   looking guards.''')

def start():
   print("\n ----------")
   print("Do you approach the...")
   print("\n")
   print("1. Ragged looking man")
   print("2. Dangerous looking guards")

   cmdlist=["1", "2"]
   cmd=getcmd(cmdlist)

   if cmd == “1”:
      ragged()
   elif cmd == “2”:
      guards()

def ragged():
   print(“\n” * 200)
   print(‘’’You walk up to the ragged looking man and
   greet him.
      He smiles a toothless grin and, with a strange
      accent, says.
      “Buy me a cup of wine, and I’ll tell you of
      great treasure...’’’)
   time.sleep(2)

def guards():
   print(“\n” *200)
   print(‘’’You walk up to the dangerous looking guards
   and greet them.
      The guards look up from their drinks and
      snarl at you.
      “What do you want, barbarian?” One guard reaches
      for the hilt of his sword...’’’)
   time.sleep(2)
```

```
def getcmd(cmdlist):
   cmd = input(name+”>”)
   if cmd in cmdlist:
      return cmd
   elif cmd == “help”:
      print(“\nEnter your choices as detailed in
      the game.”)
      print(“or enter ‘quit’ to leave the game”)
      return getcmd(cmdlist)
   elif cmd == “quit”:
      print(“\n-----------”)
      time.sleep(1)
      print(“Sadly you return to your homeland without
      fame or fortune...”)
      time.sleep(5)
      exit()


if _ _ name _ _ ==” _ _ main _ _ ”:
   start()
```

Adventure Time

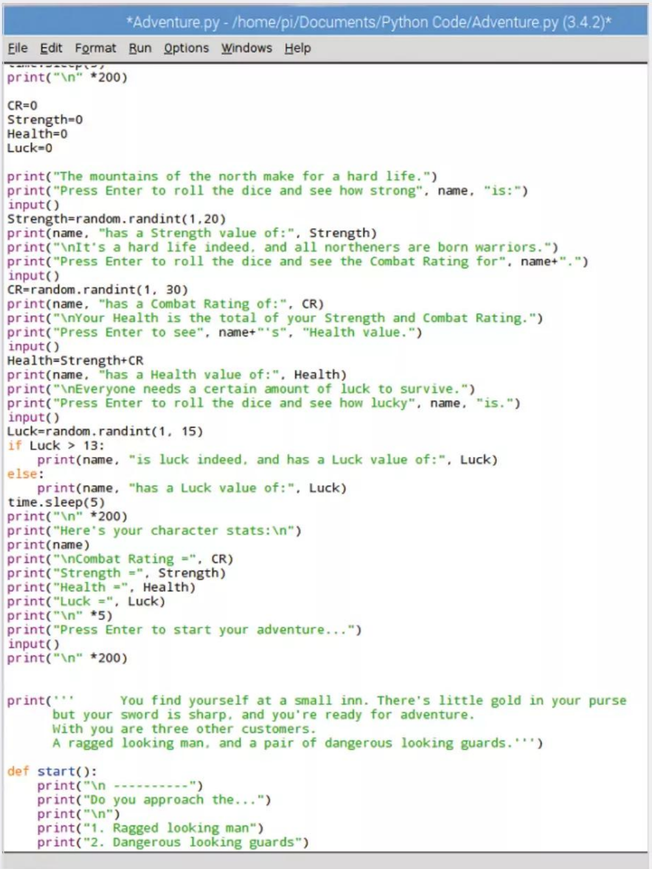
This, as you can see, is just the beginning of the adventure and takes up a fair few lines of code. When you expand it, and weave the story along, you’ll find that you can repeat certain instances such as a chance meeting with an enemy or the like.

We’ve created each of the two encounters as a defined set of functions, along with a list of possible choices under the cmdlist list, and cmd variable, of which is also a defined function. Expanding on this is quite easy, just map out each encounter and choice and create a defined function around it. Providing the user doesn’t enter quit into the adventure, they can keep playing.

There’s also room in the adventure for a set of variables designed for combat, luck, health, endurance and even an inventory or amount of gold earned. Each successful combat situation can reduce the main character’s health but increase their combat skills or endurance. Plus, they could loot the body and gain gold, or earn gold through quests.

Finally, how about introducing the Random module. This will enable you to include an element of chance in the game. For example, in combat, when you strike an enemy you will do a random amount of damage as will they. You could even work out the maths behind improving the chance of a better hit based on your or your opponent’s combat skills, current health, strength and endurance. You could create a game of dice in the inn, to see if you win or lose gold (again, improve the chances of winning by working out your luck factor into the equation).

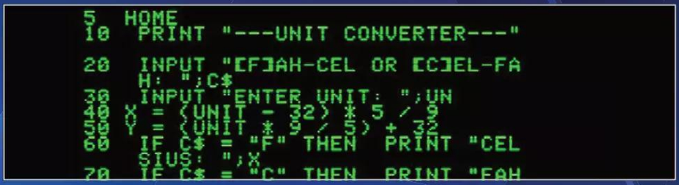
Needless to say, your text adventure can grow exponentially and prove to be a work of wonder. Good luck, and have fun with your adventure.


Adventure.py - /home/pi/Documents/Python Code/Adventure.py (3.4.2)
File Edit Format Run Options Windows Help

```
print("\n" *200)

CR=0
Strength=0
Health=0
Luck=0

print("The mountains of the north make for a hard life.")
print("Press Enter to roll the dice and see how strong", name, "is:")
input()
Strength=random.randint(1,20)
print(name, "has a Strength value of:", Strength)
print("\nIt's a hard life indeed, and all northeners are born warriors.")
print("Press Enter to roll the dice and see the Combat Rating for", name+".")
input()
CR=random.randint(1, 30)
print(name, "has a Combat Rating of:", CR)
print("\nYour Health is the total of your Strength and Combat Rating.")
print("Press Enter to see", name+"'s", "Health value.")
input()
Health=Strength+CR
print(name, "has a Health value of:", Health)
print("\nEveryone needs a certain amount of luck to survive.")
print("Press Enter to roll the dice and see how lucky", name, "is.")
input()
Luck=random.randint(1, 15)
if Luck > 13:
    print(name, "is luck indeed, and has a Luck value of:", Luck)
else:
    print(name, "has a Luck value of:", Luck)
time.sleep(5)
print("\n" *200)
print("Here's your character stats:\n")
print(name)
print("\nCombat Rating =", CR)
print("Strength =", Strength)
print("Health =", Health)
print("Luck =", Luck)
print("\n" *5)
print("Press Enter to start your adventure...")
input()
print("\n" *200)


print('''      You find yourself at a small inn. There's little gold in your purse
      but your sword is sharp, and you're ready for adventure.
      With you are three other customers.
      A ragged looking man, and a pair of dangerous looking guards.''')

def start():
    print("\n ----------")
    print("Do you approach the...")
    print("\n")
    print("1. Ragged looking man")
    print("2. Dangerous looking guards")
```

Retro Coding

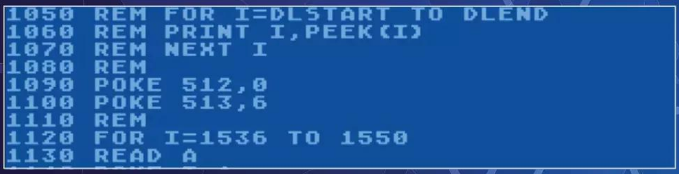
There's a school of thought, that to master the foundations of good coding skills you need to have some experience of how code was written in the past. In the past is a bit of a loose term, but mostly, it means coding from the 80s.



```
5  HOME
10  PRINT "---UNIT CONVERTER---"
20  INPUT "[F]AH-CEL OR [C]EL-FA
    H: ";C$
30  INPUT "ENTER UNIT: ";UN
40 X = (UNIT - 32) * 5 / 9
50 Y = (UNIT * 9 / 5) + 32
60  IF C$ = "F" THEN  PRINT "CEL
    SIUS: ";X
70  IF C$ = "C" THEN  PRINT "FAH
```

THE GOLDEN ERA OF CODE

While it may seem a little counterproductive to learn how to code in a language that's virtually obsolete, there are some surprising benefits to getting your hands dirty with a bit of retro coding. Firstly, learning old code will help you build the structure of code as, regardless of whether it is a language that was developed yesterday, or forty years ago, code still demands strict discipline to work correctly. Secondly, everyday coding elements, such as loops, sub routines and so on, are a great visual aid to learn in older code, especially BASIC. Lastly, it's simply good fun.



```
1050 REM FOR I=DLSTART TO DLEND
1060 REM PRINT I,PEEK(I)
1070 REM NEXT I
1080 REM
1090 POKE 512,0
1100 POKE 513,6
1110 REM
1120 FOR I=1536 TO 1550
1130 READ A
```

GOING BASIC

The easiest retro language to play around with is, without doubt, BASIC. Developed back in the mid-sixties, BASIC (Beginner's All-Purpose Symbolic Instruction Code) is a high-level programming language whose design was geared toward ease of use. In a time when computers were beginning to become more accessible, designers John Kemeny and Thomas Kurtz needed a language that students could get to grips with, quickly and easily. Think of BASIC as a distant relation to Python.

THE BEEB

The problem with BASIC is that there were so many different versions available, across multiple 8-bit platforms, with each having its own unique elements on top of the core BASIC functions. The BASIC that was packaged with the Commodore 64 was different to that on the ZX Spectrum, or the Atari home computers, due to the differing hardware of each system. However, it's widely recognised that one of the 'best', and possibly most utilised, form of BASIC from the 80s was that of BBC BASIC.

BBC BASIC was used on the Acorn BBC Micro range of computers, utilising the MOS 6502-based processor technologies. It was one of the quickest examples of BASIC and, thanks to an inline assembler, it was also capable of allowing the developers of the time to write code for different processor types, such as the Zilog Z80 – a CPU present in the ZX Spectrum, as well as many arcade machines.

The BBC Micro was designed and built by Acorn Computers – a company that is historically responsible for the creation of the ARM CPU - the processor that's used in virtually every Android phone and tablet, smart TV, set top box and so on, as well as the Raspberry Pi – so essentially, the BBC Micro is the grandfather of the Raspberry Pi.

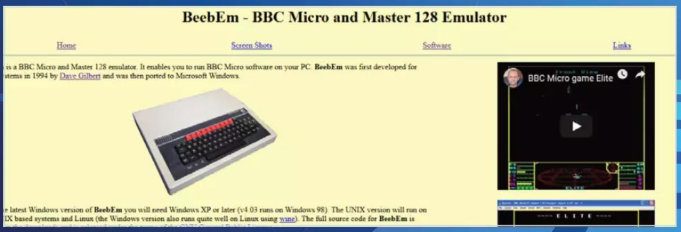
The BBC Micro was born in a time when the UK government was looking for a countrywide computer platform to be used throughout education. Different companies bid, but it was the BBC's Computer Literacy Project (the BBC Micro) that was chosen, due to its ruggedness, upgradability, and potential for education. As a result, the BBC Micro, or the Beeb as it's affectionately known, became the dominant educational computer throughout the 80s.

BEEBEM

Naturally, you could scour eBay and look for a working BBC Micro to play around on, and it'll be a lot of fun. However, for the sake of just getting hands-on with some retro code, we'll use one of the best BBC Micro emulators available: BeebEm.

BeebEm was originally developed for UNIX in 1994 by Dave Gilbert and later ported to Windows. It is now developed by Mike Wyatt and Jon Welch, who maintain the Mac port of the emulator, and is therefore available for Windows 10, Linux and macOS, as well as other platforms.

If you're using Windows 10, simply navigate to **http://www.mkw.me.uk/beebem/index.html**, and download the BeebEM414.exe that's displayed in the main screen.



BeebEm - BBC Micro and Master 128 Emulator

is a BBC Micro and Master 128 emulator. It enables you to run BBC Micro software on your PC. **BeebEm** was first developed for stems in 1994 by Dave Gilbert and was then ported to Microsoft Windows.

e latest Windows version of **BeebEm** you will need Windows XP or later (v4.03 runs on Windows 98). The UNIX version will run on IX based systems and Linux (the Windows version also runs quite well on Linux using wine). The full source code for **BeebEm** is

Once downloaded, launch the executable and follow the on-screen instructions to install it. MacOS users can get everything they need from: **http://www.g7jjf.com/**. However, Raspberry Pi and Linux users will have to do a little nifty keyboard work before they can enjoy the benefits of the Beeb on their screens. Here's how to get it working under Linux:

First, drop to a Terminal and enter:

```
sudo apt-get update && upgrade
wget http://beebem-unix.bbcmicro.com/download/
beebem-0.0.13.tar.gz
```

Then, extract the compressed files with:

```
tar zxvf beebem-0.0.13.tar.gz, then enter the newly
created Beebem folder with: cd beebem-0.0.13/.
```

Now enter the following lines, hitting Enter and answering Y to accept any changes:

```
sudo apt-get install libgtk2.0-dev libsdl1.2-dev
./configure
make
sudo make install
```

This may take some time, but when it's all done, simply enter: `beebem`, to start the BBC Micro emulator.

BBC BASIC

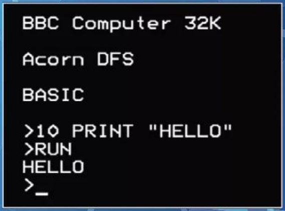
Once installed and powered up, BeebEm will display the default BBC system start-up, along with a couple of beeps. Those of you old enough to have been in a UK school in the 80s will certainly recall this setup.

In BASIC, we use line numbers to determine which lines of code run in sequence. For example, to print something to screen we'd enter:

```
10 print "hello"
```

Once you've typed the above in, press Enter and then type:

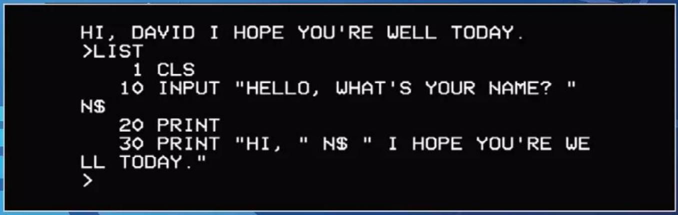
```
run
```



```
BBC Computer 32K

Acorn DFS

BASIC

>10 PRINT "HELLO"
>RUN
HELLO
>_
```

We can of course expand the code to include variables, multi-line print statements, and so on:

```
1 CLS
10 Input "Hello, what's your name? " n$
20 print
30 print "Hi, " n$ " I hope you're well today."
```



```
HI, DAVID I HOPE YOU'RE WELL TODAY.
>LIST
    1 CLS
   10 INPUT "HELLO, WHAT'S YOUR NAME? "
N$
   20 PRINT
   30 PRINT "HI, " N$ " I HOPE YOU'RE WE
LL TODAY."
>
```

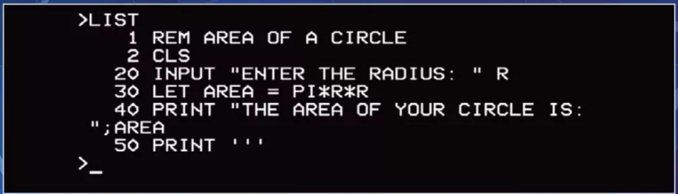
Type RUN to execute the code, you can also type LIST to view the code you've entered.

As you can see, variables are handled in much the same way as Python, a print statement on its own displays a blank line, and CLS clears the screen; although the Pi uses Clear instead of CLS. We're also able to do some maths work, and play around with variables too:

```
1 CLS
10 input "how old are you? " a
20 print
30 if a > 40 print "You're over 40 years old."
40 if a < 40 print "You're under 40 years old."
50 print
```

BBC BASIC also has some interesting built-in features, such as the value of PI:

```
1 REM Area of a circle
2 CLS
20 Input "Enter the radius: " r
30 let area = PI*r*r
40 print "The area of your circle is: "; area
50 print ''
```



```
>LIST
    1 REM AREA OF A CIRCLE
    2 CLS
   20 INPUT "ENTER THE RADIUS: " R
   30 LET AREA = PI*R*R
   40 PRINT "THE AREA OF YOUR CIRCLE IS:
 ";AREA
   50 PRINT ''
>_
```

As you'll notice, variables with a dollar ($) represent strings, nothing after the variable, or a hash (#) represent floating point decimals; a whole integer has a % character, and a byte has an ampersand (&). The single quotes after the Print on line 50 indicate a blank line, one for each tick, while REM on line 1 is a comment, and thus ignored by the BASIC compiler.

Needless to say, there's a lot you can learn, as well as having fun, with BBC BASIC. It's a rainy day project and something that's interesting to show the kids – this is how we rolled back in the 80s, kids!

There are a number of sites you can visit to learn BBC BASIC, such as **http://archive.retro-kit.co.uk/bbc.nvg.org/docs.php3.html**. See what you can come up with using BBC BASIC, or other BASIC types for different systems, and let us know what you've created.

OTHER SYSTEMS

Naturally, you don't have to look to the BBC Micro to play around with some retro code. If you grew up with a Commodore 64, then you can always try VICE, the C64 emulator. Likewise, the ZX Spectrum has a slew of great emulators available for every modern system to play around on. In fact, you can probably find an emulator for virtually every 8-bit or 16-bit machine that was produced over the years. Each has their own unique perspective and coding nuances, so find a few and see what you can create.

128
Tape Loader
128 BASIC
Calculator
48 BASIC
Tape Tester
© 1986 Sinclair Resear
Amstrad 64K Microcomputer (v1)
©1984 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.
BASIC 1.0
Ready
**** COMMODORE 64 BASIC V2 ****
GEM, Graphics Environment Manager
TOS
ATARI
Copyright (c) 1985
Atari Corp.
Digital Research, Inc.
All Rights Reserved.
OK

Text Animations

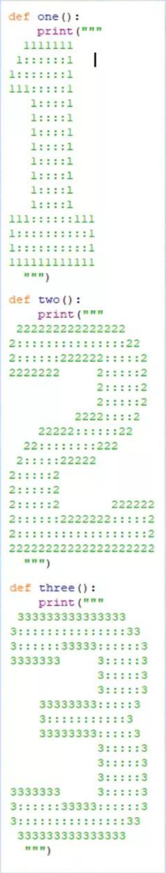
There's a remarkable amount you can do with some simple text and a little Python know-how. Combining what you've already learned, we can create some interesting animation effects from the command line.

THE FINAL COUNTDOWN

Let's begin with some example code that will display a large countdown from ten, then clear the screen and display a message. The code itself is quite simple, but lengthy. You will need to start by importing the OS and Time modules, then start creating functions that display the numbers (see image below) and so on to 10. It'll take some time, but it's worth it in the end. Of course, you can always take a different approach and design the numbers yourself.

The next step of the process is to initialise the code settings and start the countdown:

```
#Initialise settings
start = 10
message = “>        BLAST OFF!!
<”

#Start the countdown
for counter in range(start, 0, -1):
    if counter == 10:
        ten()
    elif counter == 9:
        nine()
    elif counter == 8:
        eight()
    elif counter == 7:
        seven()
    elif counter == 6:
        six()
    elif counter == 5:
        five()
    elif counter == 4:
        four()
    elif counter == 3:
        three()
    elif counter == 2:
        two()
    elif counter == 1:
        one()
    time.sleep(1)
    os.system(‘cls’ if os.name ==
‘nt’ else ‘clear’)
```

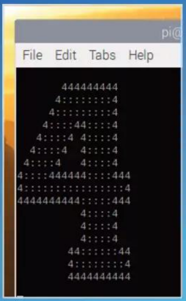


```
def one():
    print("""
  1111111
 1::::::1
1:::::::1
111:::::1
   1::::1
   1::::1
   1::::1
   1::::1
   1::::1
   1::::1
   1::::1
   1::::1
111::::::111
1::::::::::1
1::::::::::1
111111111111
  """)

def two():
    print("""
 222222222222222
2:::::::::::::::22
2::::::222222:::::2
2222222     2:::::2
            2:::::2
            2:::::2
         2222::::2
    22222::::::22
  22::::::::222
 2:::::22222
2:::::2
2:::::2
2:::::2       222222
2::::::2222222:::::2
2::::::::::::::::::2
22222222222222222222
  """)

def three():
    print("""
 333333333333333
3:::::::::::::::33
3::::::33333::::::3
3333333     3:::::3
            3:::::3
            3:::::3
    33333333:::::3
    3:::::::::::3
    33333333:::::3
            3:::::3
            3:::::3
            3:::::3
3333333     3:::::3
3::::::33333::::::3
3:::::::::::::::33
 333333333333333
  """)
```

And finally, we can add a display for the message:

```
#Display the message
print(“v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v”)
print(“<                              >”)
print(message)
print(“<                              >”)
print(“v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v”)
print(“\n\n\n”)
```

The code in its entirety can be viewed from within our Code Repository: **https://bdmpublications.com/code-portal**, where you're free to copy it to your own Python IDLE and use it as you see fit. The end effect is quite good and it'll be worth adding to your own games, or presentations, in Python.



```
       444444444
      4::::::::4
     4:::::::::4
    4::::44::::4
   4::::4 4::::4
  4::::4  4::::4
 4::::4   4::::4
4::::444444::::444
4::::::::::::::::4
4444444444:::::444
          4::::4
          4::::4
          4::::4
        44::::::44
        4::::::::4
        4444444444
```

To extend the code, or make it easier to use, you can always create the number functions in their own Python code, call it Count.py for example, then import Count at the beginning of a new Python file called **Countdown.py**, along with the OS and Time modules:

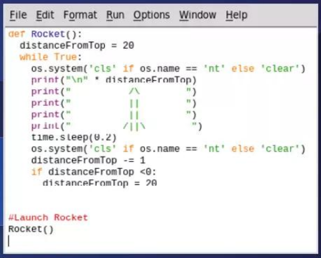
```
import os
import time
import count
```

From there, you will need to specify the imported code in the Countdown section:

```
#Start the countdown
for counter in range(start, 0, -1):
    if counter == 10:
        count.ten()
    elif counter == 9:
        count.nine()
    elif counter == 8:
        count.eight()
    elif counter == 7:
        count.seven()
    elif counter == 6:
        count.six()
    elif counter == 5:
        count.five()
    elif counter == 4:
        count.four()
    elif counter == 3:
        count.three()
    elif counter == 2:
        count.two()
    elif counter == 1:
        count.one()
```

ROCKET LAUNCH

Building on the previous countdown example, we can create an animated rocket that'll launch after the Blast Off!! message has been printed. The code would look something like:



```
def Rocket():
  distanceFromTop = 20
  while True:
    os.system('cls' if os.name == 'nt' else 'clear')
    print("\n" * distanceFromTop)
    print("          /\         ")
    print("          ||         ")
    print("          ||         ")
    print("         /||\         ")
    time.sleep(0.2)
    os.system('cls' if os.name == 'nt' else 'clear')
    distanceFromTop -= 1
    if distanceFromTop <0:
      distanceFromTop = 20


#Launch Rocket
Rocket()
```

```
def Rocket():
  distanceFromTop = 20
  while True:
    os.system(‘cls’ if os.name == ‘nt’ else ‘clear’)
    print(“\n” * distanceFromTop)
    print(“          /\          ”)
    print(“          ||          ”)
    print(“          ||          ”)
    print(“         /||\         ”)
    time.sleep(0.2)
    os.system(‘cls’ if os.name == ‘nt’ else ‘clear’)
    distanceFromTop -= 1
    if distanceFromTop <0:
      distanceFromTop = 20

#Launch Rocket
Rocket()
```

Here, we've created a new function called Rocket, which produces the effect of an ASCII-like rocket taking off and scrolling upwards; using the distanceFromTop variable.

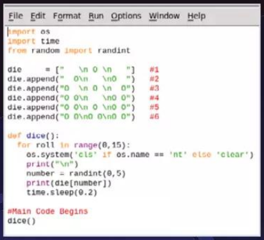
To use this, add it to the end of the previous countdown code and, at the end of the Blast Off!! message, add the following lines:

```
print(“\n\n\n\n”)
input(“Press Enter to launch rocket...”
```

This will allow your message to be displayed and then, when the user has hit the Enter button, the rocket will launch.

Again, the code in its entirety can be found in the Code Repository at: **https://bdmpublications.com/ code-portal**.

ROLLING DIE



```
import os
import time
from random import randint

die      = ["   \n O \n    "]   #1
die.append("  O\n   \nO  ")   #2
die.append("O  \n O \n  O")   #3
die.append("O O\n   \nO O")   #4
die.append("O O\n O \nO O")   #5
die.append("O O\nO O\nO O")   #6

def dice():
  for roll in range(0,15):
    os.system('cls' if os.name == 'nt' else 'clear')
    print("\n")
    number = randint(0,5)
    print(die[number])
    time.sleep(0.2)

#Main Code Begins
dice()
```

Aside from the rocket animation, together with its countdown, another fun bit of text-based animation is that of a rolling dice.

A rolling dice can be a great animation to include in an adventure game, where the player rolls to see what their score is compared to that of an enemy. The highest roller wins the round and the losers' health drops as a result. It's an age-old combat sequence, used mainly in the Dungeon and Dragons board games and Fighting Fantasy novels, but it works well.

The code you'll need to animate a dice roll is:

```
import os
import time
from random import randint

die      = [“   \n O \n   ”]   #1
die.append(“  O\n   \nO  ”)   #2
die.append(“O  \n O \n  O”)   #3
die.append(“O O\n   \nO O”)   #4
die.append(“O O\n O \nO O”)   #5
die.append(“O O\nO O\nO O”)   #6

def dice():
  for roll in range(0,15):
    os.system(‘cls’ if os.name == ‘nt’ else ‘clear’)
    print(“\n”)
    number = randint(0,5)
    print(die[number])
    time.sleep(0.2)

#Main Code Begins
dice()
```

You may need to tweak the O entries, to line up the dots on the virtual dice. Once it's done, though, you'll be able to add this function to your adventure game code and call it up whenever your character, or the situation, requires some element of luck, combat, or chance roll of the dice.

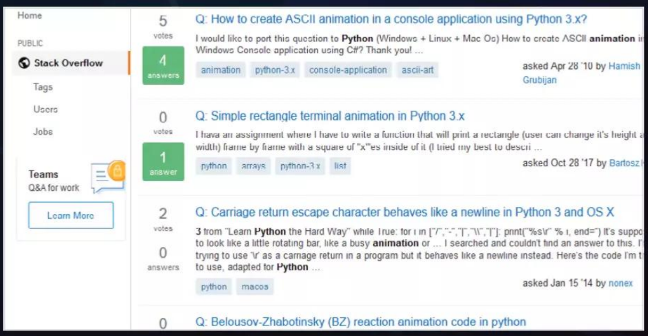
DISCOVER ANIMATIONS

The great thing about Python code is that it's so accessible. These few examples will help you add some fun, or something different, to your programs, but they're just the tip of the proverbial iceberg. If there's something you want to include in your code, and you're at a sticking point, then consider heading over to Stack Overflow and search for Python 3 content.

Stack Overflow is a great online help and resource portal, where you can ask questions and experts will try and help you. It doesn't always work out, but most of the time, you'll find what you're looking for within this great resource.

Also, if you're after a simple animation then take to Google and spend some time searching for it. While you may not find exactly what it is you're after, you're bound to come across something very like the desired effect; all you need to do is modify it slightly to accomplish your own goals.

You'll find that many professionals will often take to the Internet to find content they're after. True, they're able to code it themselves, but even experts get stuck sometimes, so don't worry about hunting code snippets down; you're in good company.



5 votes 4 answers

Q: How to create ASCII animation in a console application using Python 3.x?

I would like to port this question to **Python** (Windows + Linux + Mac Os) How to create ASCII **animation** i
Windows Console application using C#? Thank you! ...

animation python-3.x console-application ascii-art

asked Apr 28 '10 by Hamish Grubijan

0 votes 1 answer

Q: Simple rectangle terminal animation in Python 3.x

I have an assignment where I have to write a function that will print a rectangle (user can change it's height a
width) frame by frame with a square of "x"'es inside of it (I tried my best to descri ...

python arrays python-3.x list

asked Oct 28 '17 by Bartosz

2 votes 0 answers

Q: Carriage return escape character behaves like a newline in Python 3 and OS X

3 from "Learn **Python** the Hard Way" while True: for i in ["/","-","|","\\","|"]: print("%s\r" % i, end='') It's suppo
to look like a little rotating bar, like a busy **animation** or ... I searched and couldn't find an answer to this. I'
trying to use '\r' as a carriage return in a program but it behaves like a newline instead. Here's the code I'm t
to use, adapted for **Python** ...

python macos

asked Jan 15 '14 by nonex

0

Q: Belousov-Zhabotinsky (BZ) reaction animation code in python

Common Coding Mistakes

When you start something new you're inevitably going to make mistakes, this is purely down to inexperience and those mistakes are great teachers in themselves. However, even experts make the occasional mishap. Thing is, to learn from them as best you can.

X=MISTAKE, PRINT Y

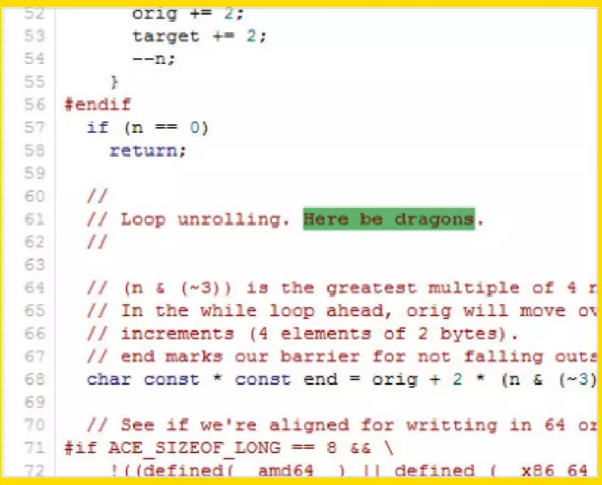
There are many pitfalls for the programmer to be aware of, far too many to be listed here. Being able to recognise a mistake and fix it is when you start to move into more advanced territory.

SMALL CHUNKS

It would be wonderful to be able to work like Neo from The Matrix movies. Simply ask, your operator loads it into your memory and you instantly know everything about the subject. Sadly though, we can't do that. The first major pitfall is someone trying to learn too much, too quickly. So take coding in small pieces and take your time.

//COMMENTS

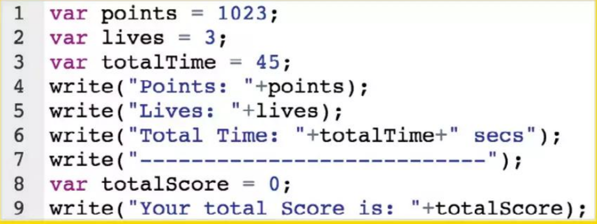
Use comments. It's a simple concept but commenting on your code saves so many problems when you next come to look over it. Inserting comment lines helps you quickly sift through the sections of code that are causing problems; also useful if you need to review an older piece of code.



```
        orig += 2;
        target += 2;
        --n;
    }
#endif
  if (n == 0)
    return;

  //
  // Loop unrolling. Here be dragons.
  //

  // (n & (~3)) is the greatest multiple of 4 r
  // In the while loop ahead, orig will move ov
  // increments (4 elements of 2 bytes).
  // end marks our barrier for not falling outs
  char const * const end = orig + 2 * (n & (~3)

  // See if we're aligned for writting in 64 or
#if ACE_SIZEOF_LONG == 8 && \
      !((defined(  amd64  ) || defined (  x86_64
```

EASY VARIABLES

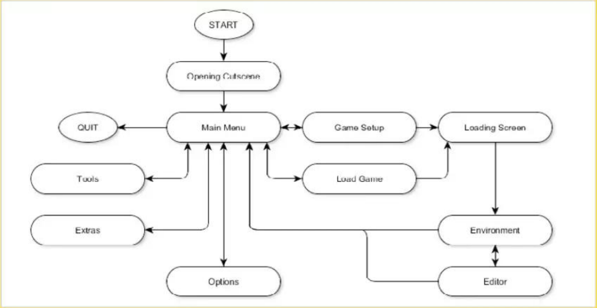
Meaningful naming for variables is a must to eliminate common coding mistakes. Having letters of the alphabet is fine but what happens when the code states there's a problem with x variable. It's not too difficult to name variables lives, money, player1 and so on.



```
var points = 1023;
var lives = 3;
var totalTime = 45;
write("Points: "+points);
write("Lives: "+lives);
write("Total Time: "+totalTime+" secs");
write("--------------------------");
var totalScore = 0;
write("Your total Score is: "+totalScore);
```

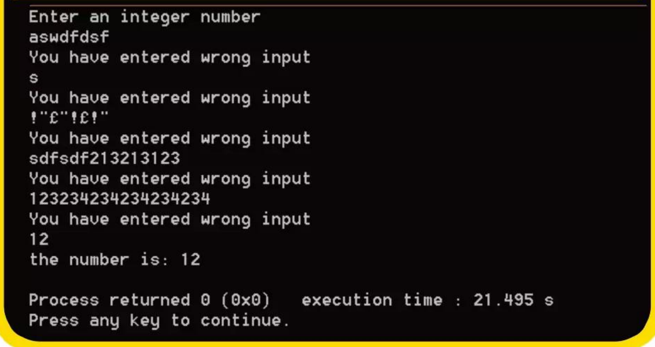
PLAN AHEAD

While it's great to wake up one morning and decide to code a classic text adventure, it's not always practical without a good plan. Small snippets of code can be written without too much thought and planning but longer and more in-depth code requires a good working plan to stick to and help iron out the bugs.



USER ERROR

User input is often a paralysing mistake in code. For example, when the user is supposed to enter a number for their age and instead they enter it in letters. Often a user can enter so much into an input that it overflows some internal buffer, thus sending the code crashing. Watch those user inputs and clearly state what's needed from them.



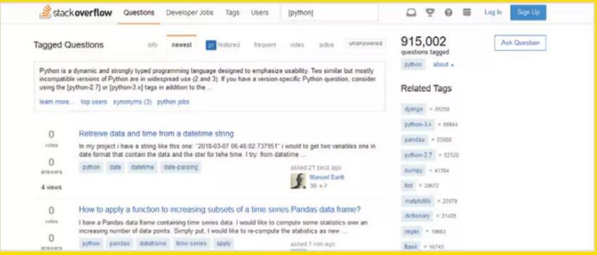
```
Enter an integer number
aswdfdsf
You have entered wrong input
s
You have entered wrong input
!"£"!£!"
You have entered wrong input
sdfsdf213213123
You have entered wrong input
1232342342342342 34
You have entered wrong input
12
the number is: 12

Process returned 0 (0x0)   execution time : 21.495 s
Press any key to continue.
```

RE-INVENTING WHEELS

You can easily spend days trying to fathom out a section of code to achieve a given result and it's frustrating and often time-wasting. While it's equally rewarding to solve the problem yourself, often the same code is out there on the Internet somewhere. Don't try and re-invent the wheel, look to see if some else has done it first.

HELP!

Asking for help is something most of us has struggled with in the past. Will the people we're asking laugh at us? Am I wasting everyone's time? It's a common mistake for someone to suffer in silence. However, as long as you ask the query in the correct manner, obey any forum rules and be polite, then your question isn't silly.



Python is a dynamic and strongly typed programming language designed to emphasize usability. Two similar but mostly incompatible versions of Python are in widespread use (2 and 3). If you have a version-specific Python question, consider using the [python-2.7] or [python-3.x] tags in addition to the ...

0 votes 0 answers 4 views

Retreive data and time from a datetime string

In my project i have a string like this one: "2018-03-07 06:46:02.737951" i would to get two variables one in date format that contain the data and the oter for tehe time. I try: from datetime ...

python date datetime date-parsing

asked 21 secs ago Manuel Sanb 30 •7

0 votes 0 answers

How to apply a function to increasing subsets of a time series Pandas data frame?

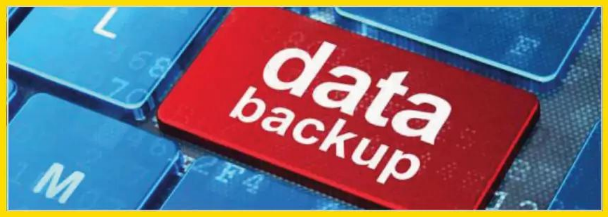
I have a Pandas data frame containing time series data. I would like to compute some statistics over an increasing number of data points. Simply put, I would like to re-compute the statistics as new ...

python pandas dataframe time-series apply

asked 1 min ago

BACKUPS

Always make a backup of your work, with a secondary backup for any changes you've made. Mistakes can be rectified if there's a good backup in place to revert to for those times when something goes wrong. It's much easier to start where you left off, rather than starting from the beginning again.



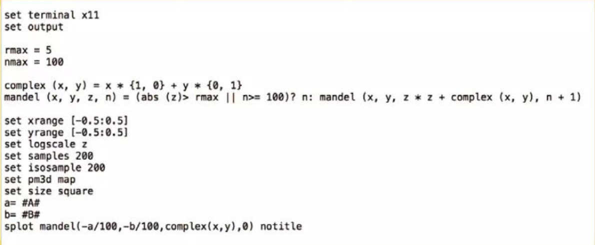
SECURE DATA

If you're writing code to deal with usernames and passwords, or other such sensitive data, then ensure that the data isn't in cleartext. Learn how to create a function to encrypt sensitive data, prior to feeding into a routine that can transmit or store it where someone may be able to get to view it.

ENCRYPT
data
secure
protection
privilege
unauthorized
hacker
software
device
computer
program

MATHS

If your code makes multiple calculations then you need to ensure that the maths behind it is sound. There are thousands of instances where programs have offered incorrect data based on poor mathematical coding, which can have disastrous effects depending on what the code is set to do. In short, double check your code equations.



```
set terminal x11
set output

rmax = 5
nmax = 100

complex (x, y) = x * {1, 0} + y * {0, 1}
mandel (x, y, z, n) = (abs (z)> rmax || n>= 100)? n: mandel (x, y, z * z + complex (x, y), n + 1)

set xrange [-0.5:0.5]
set yrange [-0.5:0.5]
set logscale z
set samples 200
set isosample 200
set pm3d map
set size square
a= #A#
b= #B#
splot mandel(-a/100,-b/100,complex(x,y),0) notitle
```

Python Beginner's Mistakes

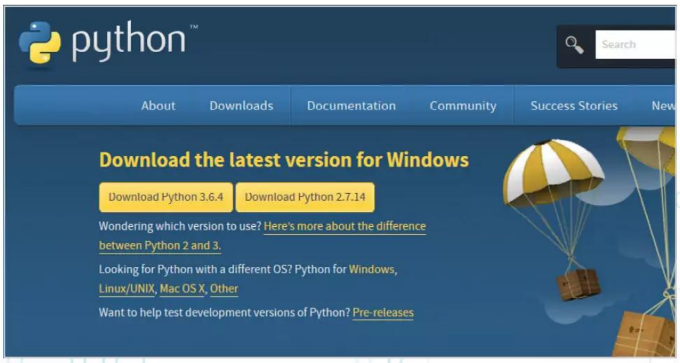
Python is a relatively easy language to get started in where there's plenty of room for the beginner to find their programming feet. However, as with any other programming language, it can be easy to make common mistakes that'll stop your code from running.

DEF BEGINNER(MISTAKES=10)

Here are ten common Python programming mistakes most beginners find themselves making. Being able to identify these mistakes will save you headaches in the future.

VERSIONS

To add to the confusion that most beginners already face when coming into programming, Python has two live versions of its language available to download and use. There is Python version 2.7.x and Python 3.6.x. The 3.6.x version is the most recent, and the one we'd recommend starting. But, version 2.7.x code doesn't always work with 3.6.x code and vice versa.



Download the latest version for Windows

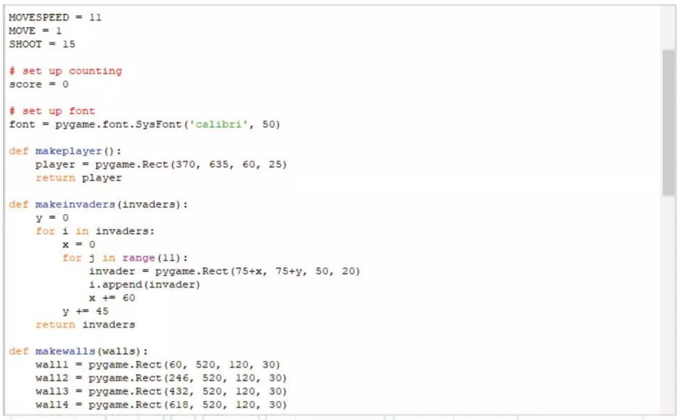
Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

INDENTS, TABS AND SPACES

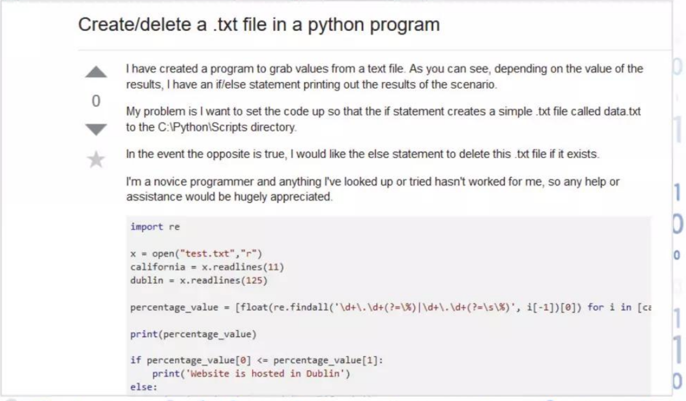
Python uses precise indentations when displaying its code. The indents mean that the code in that section is a part of the previous statement, and not something linked with another part of the code. Use four spaces to create an indent, not the Tab key.



```
MOVESPEED = 11
MOVE = 1
SHOOT = 15

# set up counting
score = 0

# set up font
font = pygame.font.SysFont('calibri', 50)

def makeplayer():
    player = pygame.Rect(370, 635, 60, 25)
    return player

def makeinvaders(invaders):
    y = 0
    for i in invaders:
        x = 0
        for j in range(11):
            invader = pygame.Rect(75+x, 75+y, 50, 20)
            i.append(invader)
            x += 60
        y += 45
    return invaders

def makewalls(walls):
    wall1 = pygame.Rect(60, 520, 120, 30)
    wall2 = pygame.Rect(246, 520, 120, 30)
    wall3 = pygame.Rect(432, 520, 120, 30)
    wall4 = pygame.Rect(618, 520, 120, 30)
```

THE INTERNET

Every programmer has and does at some point go on the Internet and copy some code to insert into their own routines. There's nothing wrong with using others' code, but you need to know how the code works and what it does before you go blindly running it on your own computer.



Create/delete a .txt file in a python program

I have created a program to grab values from a text file. As you can see, depending on the value of the results, I have an if/else statement printing out the results of the scenario.

My problem is I want to set the code up so that the if statement creates a simple .txt file called data.txt to the C:\Python\Scripts directory.

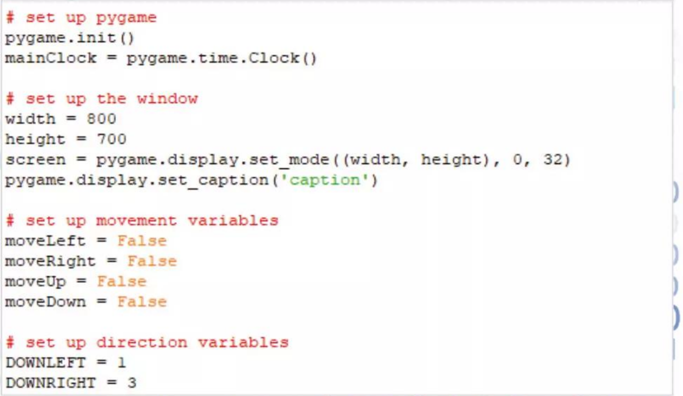
In the event the opposite is true, I would like the else statement to delete this .txt file if it exists.

I'm a novice programmer and anything I've looked up or tried hasn't worked for me, so any help or assistance would be hugely appreciated.

```
import re

x = open("test.txt","r")
california = x.readlines(11)
dublin = x.readlines(125)

percentage_value = [float(re.findall('\d+\.\d+(?=\%)|\d+\.\d+(?=\s\%)', i[-1])[0]) for i in [ca

print(percentage_value)

if percentage_value[0] <= percentage_value[1]:
    print('Website is hosted in Dublin')
else:
```

COMMENTING

Again we mention commenting. It's a hugely important factor in programming, even if you're the only one who is ever going to view the code, you need to add comments as to what's going on. Is this function where you lose a life? Write a comment and help you, or anyone else, see what's going on.



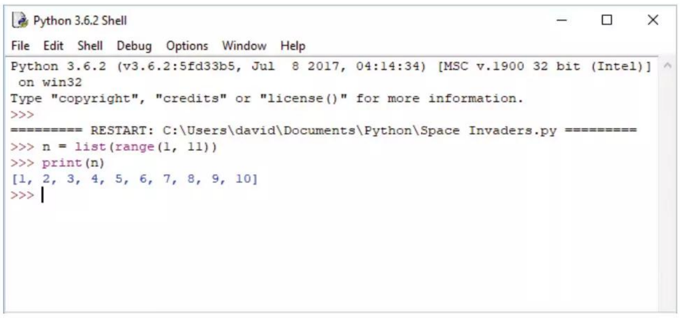
```
# set up pygame
pygame.init()
mainClock = pygame.time.Clock()

# set up the window
width = 800
height = 700
screen = pygame.display.set_mode((width, height), 0, 32)
pygame.display.set_caption('caption')

# set up movement variables
moveLeft = False
moveRight = False
moveUp = False
moveDown = False

# set up direction variables
DOWNLEFT = 1
DOWNRIGHT = 3
```

COUNTING LOOPS

Remember that in Python a loop doesn't count the last number you specify in a range. So if you wanted the loop to count from 1 to 10, then you will need to use:

`n = list(range(1, 11))`

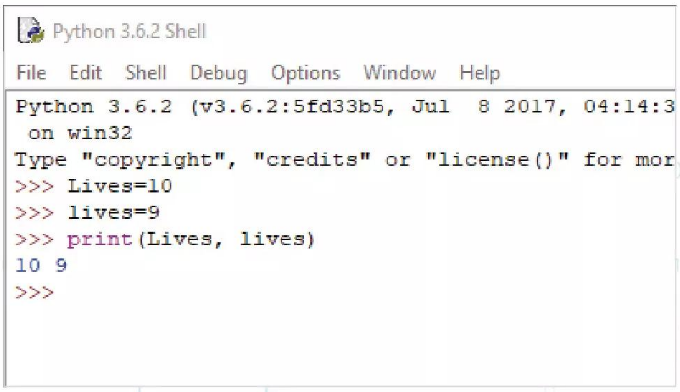
Which will return 1 to 10.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
========== RESTART: C:\Users\david\Documents\Python\Space Invaders.py ==========
>>> n = list(range(1, 11))
>>> print(n)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> 
```

CASE SENSITIVE

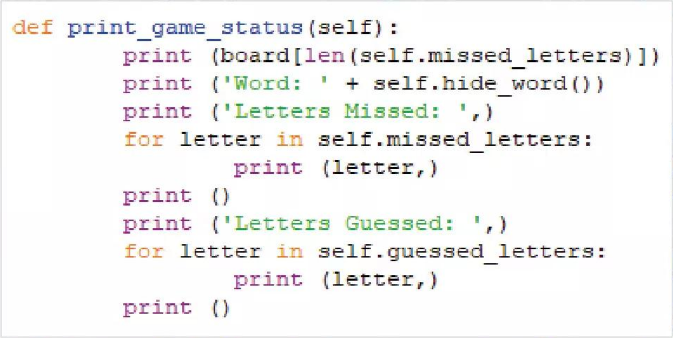
Python is a case sensitive programming language, so you will need to check any variables you assign. For example, **`Lives=10`** is a different variable to lives=10, calling the wrong variable in your code can have unexpected results.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:3
 on win32
Type "copyright", "credits" or "license()" for mor
>>> Lives=10
>>> lives=9
>>> print(Lives, lives)
10 9
>>> 
```

BRACKETS

Everyone forgets to include that extra bracket they should have added to the end of the statement. Python relies on the routine having an equal amount of closed brackets to open brackets, so any errors in your code could be due to you forgetting to count your brackets; including square brackets.



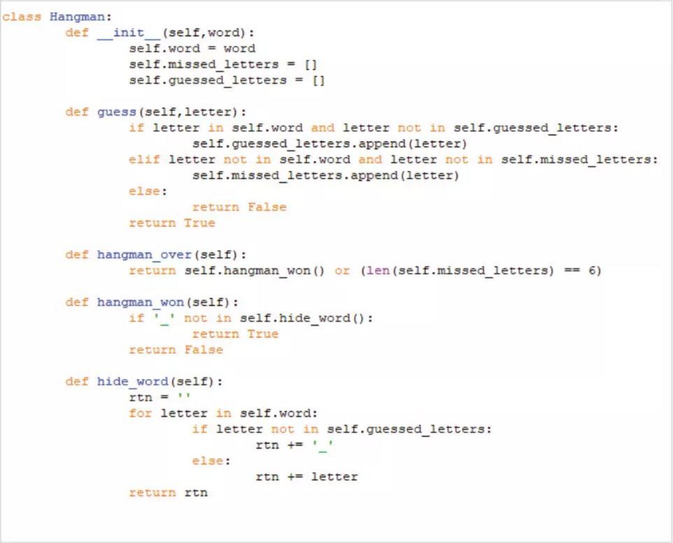
```
def print_game_status(self):
        print (board[len(self.missed_letters)])
        print ('Word: ' + self.hide_word())
        print ('Letters Missed: ',)
        for letter in self.missed_letters:
                print (letter,)
        print ()
        print ('Letters Guessed: ',)
        for letter in self.guessed_letters:
                print (letter,)
        print ()
```

COLONS

It's common for beginners to forget to add a colon to the end of a structural statement, such as:

```
class Hangman:
def guess(self, letter):
```

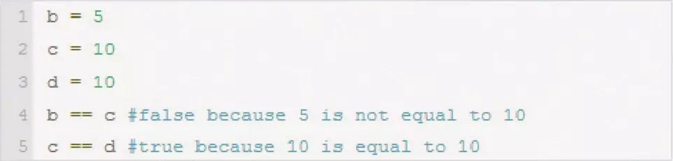
And so on. The colon is what separates the code, and creates the indents to which the following code belongs to.



```
class Hangman:
        def __init__(self,word):
                self.word = word
                self.missed_letters = []
                self.guessed_letters = []

        def guess(self,letter):
                if letter in self.word and letter not in self.guessed_letters:
                        self.guessed_letters.append(letter)
                elif letter not in self.word and letter not in self.missed_letters:
                        self.missed_letters.append(letter)
                else:
                        return False
                return True

        def hangman_over(self):
                return self.hangman_won() or (len(self.missed_letters) == 6)

        def hangman_won(self):
                if '_' not in self.hide_word():
                        return True
                return False

        def hide_word(self):
                rtn = ''
                for letter in self.word:
                        if letter not in self.guessed_letters:
                                rtn += '_'
                        else:
                                rtn += letter
                return rtn
```

OPERATORS

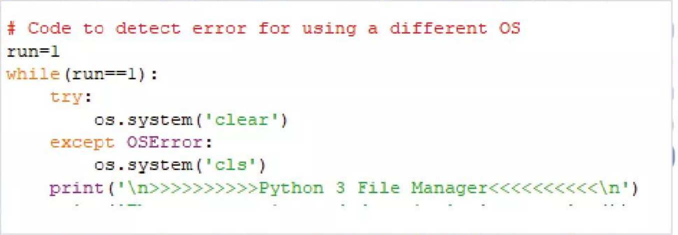
Using the wrong operator is also a common mistake to make. When you're performing a comparison between two values, for example, you need to use the equality operator (a double equals, ==). Using a single equal (=) is an assignment operator that places a value to a variable (such as, lives=10).



```
b = 5
c = 10
d = 10
b == c #false because 5 is not equal to 10
c == d #true because 10 is equal to 10
```

OPERATING SYSTEMS

Writing code for multiple platforms is difficult, especially when you start to utilise the external commands of the operating system. For example, if your code calls for the screen to be cleared, then for Windows you would use **`cls`**. Whereas, for Linux you need to use **`clear`**. You need to solve this by capturing the error and issuing it with an alternative command.



```
# Code to detect error for using a different OS
run=1
while(run==1):
    try:
        os.system('clear')
    except OSError:
        os.system('cls')
    print('\n>>>>>>>>>>Python 3 File Manager<<<<<<<<<<\n')
```

C++ Beginner’s Mistakes

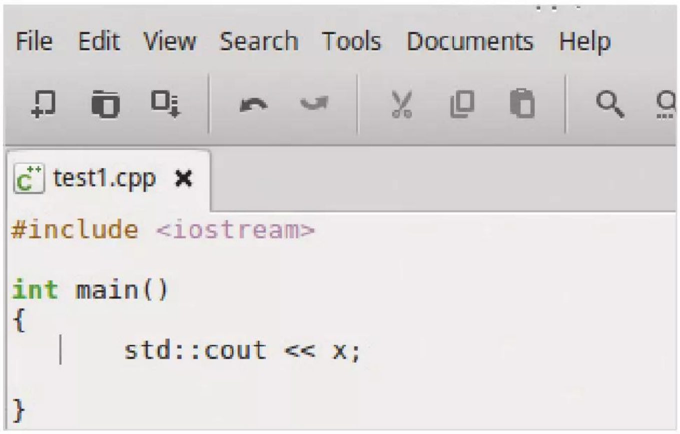
There are many pitfalls the C++ developer can encounter, especially as this is a more complex and often unforgiving language to master. Beginners need to take C++ a step at a time and digest what they’ve learned before moving on.

VOID(C++, MISTAKES)

Admittedly it’s not just C++ beginners that make the kinds of errors we outline on these pages, even hardened coders are prone to the odd mishap here and there. Here are some common issues to try and avoid.

UNDECLARED IDENTIFIERS

A common C++ mistake, and to be honest a common mistake with most programming languages, is when you try and output a variable that doesn’t exist. Displaying the value of x on-screen is fine but not if you haven’t told the compiler what the value of x is to begin with.



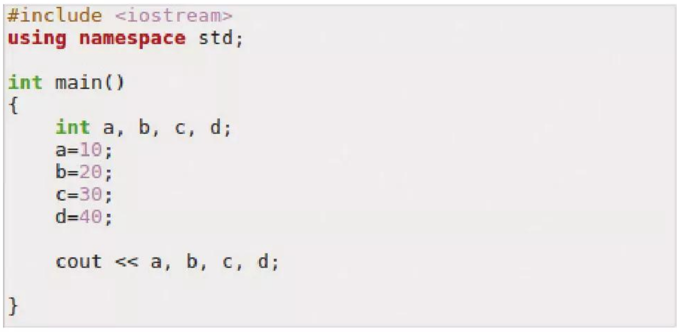
```
#include <iostream>

int main()
{
        std::cout << x;

}
```

STD NAMESPACE

Referencing the Standard Library is common for beginners throughout their code, but if you miss the std:: element of a statement, your code errors out when compiling. You can combat this by adding:

```
using namespace std;
```

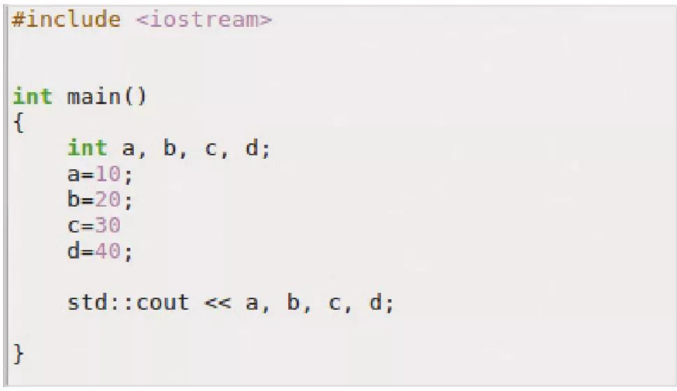
Under the #include part and simply using cout, cin and so on from then on.



```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30;
    d=40;

    cout << a, b, c, d;

}
```

SEMICOLONS

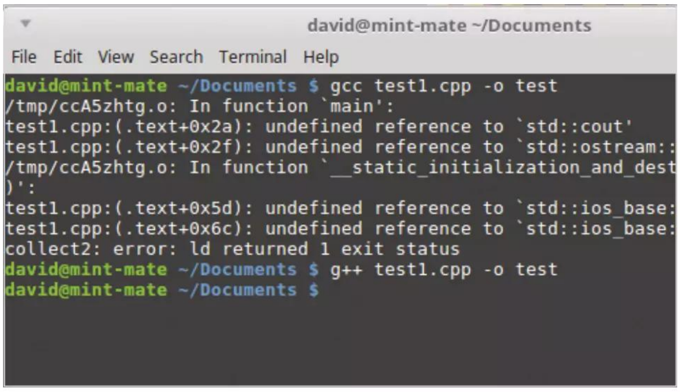
Remember that each line of a C++ program must end with a semicolon. If it doesn’t then the compiler treats the line with the missing semicolon as the same line with the next semicolon on. This creates all manner of problems when trying to compile, so don’t forget those semicolons.



```
#include <iostream>


int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30
    d=40;

    std::cout << a, b, c, d;

}
```

GCC OR G++

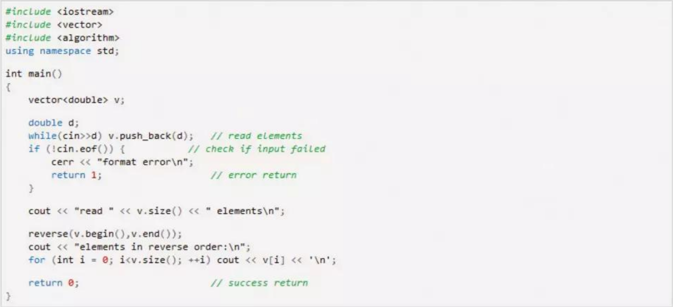
If you’re compiling in Linux then you will no doubt come across gcc and g++. In short, gcc is the Gnu Compiler Collection (or Gnu C Compiler as it used to be called) and g++ is the Gnu ++ (the C++ version) of the compiler. If you’re compiling C++ then you need to use g++, as the incorrect compiler drivers will be used.


david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ gcc test1.cpp -o test
/tmp/ccA5zhtg.o: In function `main':
test1.cpp:(.text+0x2a): undefined reference to `std::cout'
test1.cpp:(.text+0x2f): undefined reference to `std::ostream::
/tmp/ccA5zhtg.o: In function `__static_initialization_and_dest
)':
test1.cpp:(.text+0x5d): undefined reference to `std::ios_base:
test1.cpp:(.text+0x6c): undefined reference to `std::ios_base:
collect2: error: ld returned 1 exit status
david@mint-mate ~/Documents $ g++ test1.cpp -o test
david@mint-mate ~/Documents $
```

COMMENTS (AGAIN)

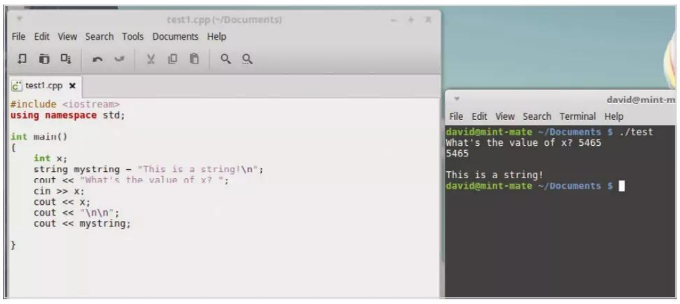
Indeed the mistake of never making any comments on code is back once more. As we've previously bemoaned, the lack of readable identifiers throughout the code makes it very difficult to look back at how it worked, for both you and someone else. Use more comments.



```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    vector<double> v;

    double d;
    while(cin>>d) v.push_back(d);   // read elements
    if (!cin.eof()) {             // check if input failed
        cerr << "format error\n";
        return 1;                   // error return
    }

    cout << "read " << v.size() << " elements\n";

    reverse(v.begin(),v.end());
    cout << "elements in reverse order:\n";
    for (int i = 0; i<v.size(); ++i) cout << v[i] << '\n';

    return 0;                       // success return
}
```

QUOTES

Missing quotes is a common mistake to make, for every level of user. Remember that quotes need to encase strings and anything that's going to be outputted to the screen or into a file, for example. Most compilers errors are due to missing quotes in the code.


test1.cpp (~/Documents)
File Edit View Search Tools Documents Help
test1.cpp
david@mint-m
File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    cout << "\n\n";
    cout << mystring;

}
```

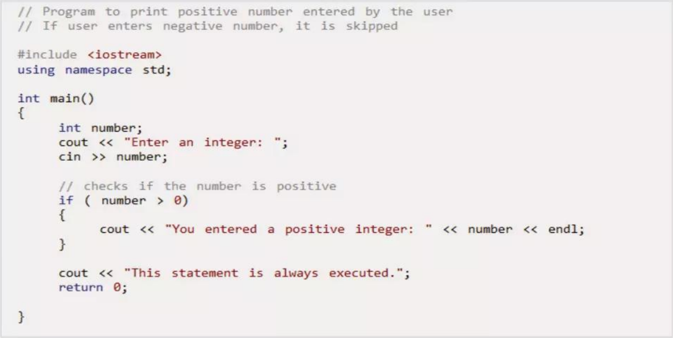
```
david@mint-mate ~/Documents $ ./test
What's the value of x? 5465
5465

This is a string!
david@mint-mate ~/Documents $
```

EXTRA SEMICOLONS

While it's necessary to have a semicolon at the end of every C++ line, there are some exceptions to the rule. Semicolons need to be at the end of every complete statement but some lines of code aren't complete statements. Such as:

```
#include
if lines
switch lines
```

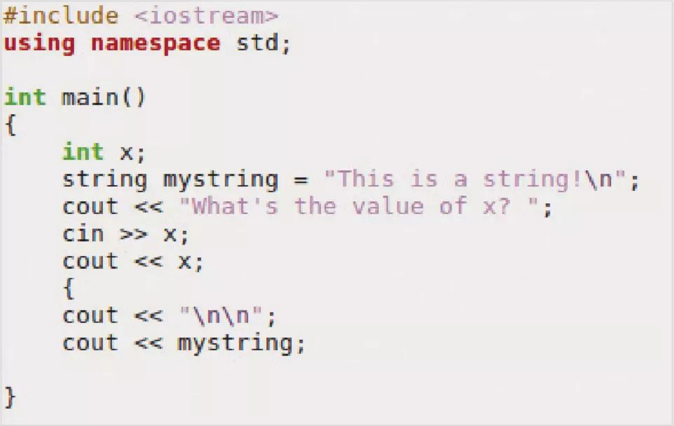
If it sounds confusing don't worry, the compiler lets you know where you went wrong.



```
// Program to print positive number entered by the user
// If user enters negative number, it is skipped

#include <iostream>
using namespace std;

int main()
{
     int number;
     cout << "Enter an integer: ";
     cin >> number;

     // checks if the number is positive
     if ( number > 0)
     {
          cout << "You entered a positive integer: " << number << endl;
     }

     cout << "This statement is always executed.";
     return 0;

}
```

TOO MANY BRACES

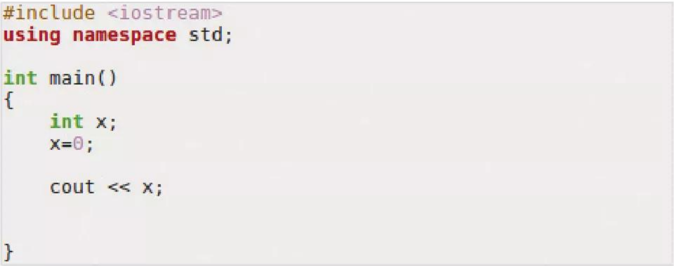
The braces, or curly brackets, are beginning and ending markers around blocks of code. So for every { you must have a }. Often it's easy to include or miss out one or the other facing brace when writing code; usually when writing in a text editor, as an IDE adds them for you.



```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    {
    cout << "\n\n";
    cout << mystring;

}
```

INITIALISE VARIABLES

In C++ variables aren't initialised to zero by default. This means if you create a variable called x then, potentially, it is given a random number from 0 to 18,446,744,073,709,551,616, which can be difficult to include in an equation. When creating a variable, give it the value of zero to begin with: `x=0`.



```
#include <iostream>
using namespace std;

int main()
{
    int x;
    x=0;

    cout << x;

}
```

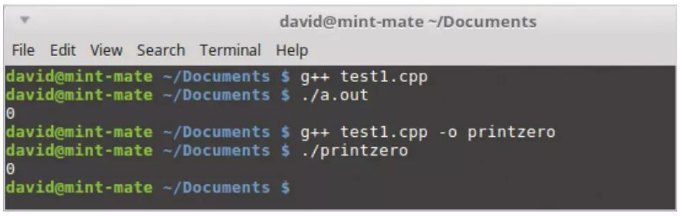
A.OUT

A common mistake when compiling in Linux is forgetting to name your C++ code post compiling. When you compile from the Terminal, you enter:

```
g++ code.cpp
```

This compiles the code in the file code.cpp and create an a.out file that can be executed with ./a.out. However, if you already have code in a.out then it's overwritten. Use:

```
g++ code.cpp -o nameofprogram
```


david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ g++ test1.cpp
david@mint-mate ~/Documents $ ./a.out
0
david@mint-mate ~/Documents $ g++ test1.cpp -o printzero
david@mint-mate ~/Documents $ ./printzero
0
david@mint-mate ~/Documents $
```

Black Dog Media

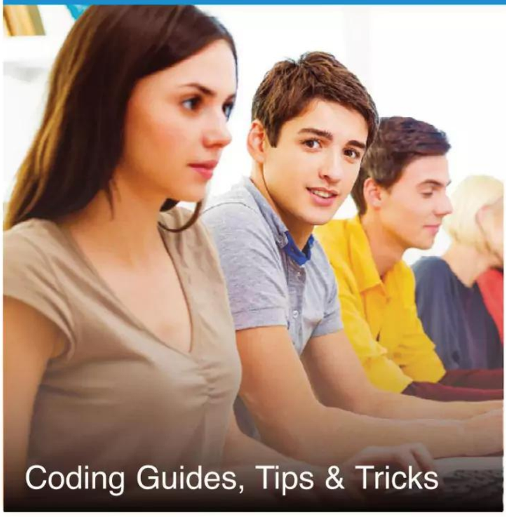
Master Your Tech

From Beginner to Expert

To continue learning more about your tech visit us at:

www.bdmpublications.com

FREE Coding Guides


Coding Guides, Tips & Tricks

Raspberry Pi 4

Programming with Python

Learn & Master C++

Testing Linux Distros

EXCLUSIVE Offers on Coding Guidebooks

- Print & digital editions
- Featuring the latest updates
- Step-by-step tutorials & guides
- Created by BDM experts

160 PAGES of PYTHON tutorials
Coding for Python
NEW
845
190 PAGES
THE COMPLETE Coding MANUAL
NEW 2020
FREE DOWNLOAD

ultimate photoshop

bdmpublications.com/ultimate-photoshop
Buy our Photoshop guides and download tutorial images for free! *Simply sign up and get creative.*

PLUS SPECIAL DEALS and Bonus Content When you sign up to our monthly newsletter!

Coding for Beginners
ISBN: 978-1-912847-91-4

Published by: Papercut Limited
Editor: James Gale
Art Director & Production: Mark Ayshford
Production Manager: Karl Linstead
Design: Robin Drew, Lena Whitaker
Editorial: David Hayward
Sub Editor: Alison Drew

Digital distribution: Readly AB, Pocketmags

Copyright © 2021 Papercut Limited. All rights reserved.

INTERNATIONAL LICENSING – Papercut Limited has many great publications and all are available for licensing worldwide. For more information go to: www.brucesawfordlicensing.com; email: bruce@brucesawfordlicensing.com; telephone: 0044 7831 567372

Editorial and design are the copyright © Papercut Limited. No part of this publication may be reproduced in any form, stored in a retrieval system, or integrated into any other publication, database, or commercial programs without the express written permission of the publisher. Under no circumstances should this publication and its contents be resold, loaned out, or used in any form by way of trade without the publisher's written permission. While we pride ourselves on the quality of the information we provide,Papercut Limited reserves the right not to be held responsible for any mistakes or inaccuracies found within the text of this publication. Due to the nature of the software industry, the publisher cannot guarantee that all tutorials will work on every version of the coding software featured. It remains the purchaser's sole responsibility to determine the suitability of this book and its content for whatever purpose. Images reproduced on the front and back covers are solely for design purposes and are not representative of content. We advise all potential buyers to check listing prior to purchase for confirmation of actual content. All editorial opinion herein is that of the reviewer as an individual and is not representative of the publisher or any of its affiliates. Therefore, the publisher holds no responsibility in regard to editorial opinion and content.

Coding for Beginners is an independent publication and as such does not necessarily reflect the views or opinions of the producers contained within. This publication is not endorsed or associated in any way with The Linux Foundation, The Raspberry Pi Foundation, ARM Holding, Canonical Ltd, Python, Debian Project, Linux Mint, Microsoft, Lenovo, Dell, Hewlett-Packard, Apple and Samsung or any associate or affiliate company. All copyrights, trademarks and registered trademarks for the respective companies are acknowledged. Relevant graphic imagery reproduced with courtesy of Lenovo, Hewlett- Packard, Dell, Microsoft, Samsung, Linux Mint, NASA, and Apple.
Additional images contained within this publication are reproduced under licence from Shutterstock.com.
Prices, international availability, ratings, titles and content are subject to change. All information was correct at time of print. Some content may have been previously published in other volumes or Papercut Limited titles. We advise potential buyers to check the suitability of contents prior to purchase.

Papercut Limited
Registered in England & Wales No: 04308513